Asistente virtual para estudiantes de la FIB

Trabajo de fin de grado

Autor:
Víctor Busqué Somacarrera

Director:
Javier Béjar Alonso
Departamento:
Ciencias de la Computación

Grado en Ingeniería informática
Especialidad en computación

Facultat d'Informàtica de Barcelona (FIB)
Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

2018

Resumen

La información acerca de la Facultad de Informática de Barcelona (FIB) está distribuida en tres portales web, por lo que los estudiantes deben saber dónde encontrar la información que necesitan para poder resolver alguna duda que tengan. La creación de un único mecanismo que permita consultar información de forma cómoda y sin necesidad de saber dónde encontrarla sería algo muy útil.

Con esta finalidad, se ha desarrollado un bot conversacional para Telegram Messenger que tiene la finalidad de actuar como asistente particular, recibiendo consultas por parte de los estudiantes, y proporcionando la respuesta. Además se ha dado la posibilidad al bot de notificar de forma automática cuando se publican avisos nuevos en el Racó.

Para llevar a cabo tal tarea, se han generado modelos basados en Machine Learning capaces de comprender consultas usando lenguaje natural en castellano, catalán e inglés. Además también se ha hecho un modelo capaz de razonar cómo responder a las consultas.

Los resultados del proyecto muestran que el bot desarrollado es capaz de reconocer las 14 posibles intenciones consideradas para el proyecto, y llevar a cabo 18 posibles acciones como respuesta. Además, es capaz de razonar si dispone de suficiente información para resolver las consultas. Finalmente, las notificaciones automáticas con los avisos del Racó son enviadas más rápidamente que el servicio de notificación que ofrece la facultad.

Resum

La informació sobre la Facultat d'Informàtica de Barcelona (FIB) està distribuïda a tres portals web, per tant, els estudiants han de saber on trobar la informació que necessiten per resoldre algun dubte que tinguin. La creació d'un únic mecanisme que permeti consultar informació de forma còmoda sense necessitat de saber on trobar-la seria quelcom molt útil.

Amb aquesta finalitat, s'ha desenvolupat un bot conversacional per a Telegram Messenger que té la finalitat d'actuar com assistent particular, rebent consultes per part dels estudiants, i proporcionant la resposta. A més, s'ha donat la possibilitat al bot de notificar de forma automàtica quan es publiquen avisos nous al Racó.

Per a dur a terme aquesta tasca, s'han generat models basats en Machine Learning capaços de comprendre preguntes fent servir llenguatge natural en castellà, català i anglès. A més, també s'ha fet un model capaç de raonar com s'han de respondre a les consultes.

Els resultats del projecte mostren que el bot desenvolupat és capaç de reconèixer les 14 possibles intencions considerades pel projecte, i dur a terme 18 possibles accions com a resposta. A més, és capaç de raonar si disposa de prou informació per a resoldre les consultes. Finalment, les notificacions automàtiques amb els avisos del Racó són enviades més ràpidament que el servei de notificació que ofereix la facultat.

Abstract

Information about Barcelona School of Informatics (FIB) is distributed among three websites, therefore, students have to know where to find the information they need in order to solve any doubt they might have. The creation of a unique mechanism that allows to consult information without knowing where to find it would be useful.

With that objective, a conversational bot for Telegram Messenger has been developed, and its aim is to act as a personal assistant for students, receiving queries from students, and providing an answer. Furthermore, the bot has been given the possibility to automatically notify when there are new publications at Racó.

In order to do that, models based on Machine Learning that can understand questions using natural language have been developed for spanish, catalan and english. Also, a model that is able to reason about how to respond to those questions has been made.

The results show that the bot is able to recognize the 14 distinct intents considered for the project, and take 18 different possible actions as a response. Moreover, the bot is able to reason if it has enough information to solve any question. Finally, the automatic notifications with Racó's new publications are sent faster than the notification service provided by the faculty.

Glosario

Asistente virtual	Agente software que ayuda a usuarios automatizando y realizando tareas mediante la interacción hombre-máquina.
Machine Learning	Rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan a las máquinas aprender.
NLP	Rama de la inteligencia artificial que estudia las interacciones entre el lenguaje humano y la máquina.
NLU	Rama del procesado de lenguaje natural (NLP) centrada en la comprensión de éste.
NER	Herramienta cuyo objetivo es reconocer entidades nombradas, por ejemplo, nombres o localizaciones.
Web Scraping	Técnica que, mediante software, extrae información de sitios web.

Glossari

Asistent virtual	Agent software que ajuda a usuaris automatitzant i realitzant tasques mitjançant la interacció home-màquina.
Machine Learning	Branca de la intel.ligència artificial que té com a objectiu desenvolupar tècniques que permetin a les màquines aprendre.
NLP	Branca de la intel.ligència artificial que estudia les interaccions entre el llenguatge humà i la màquina.
NLU	Branca del processat de llenguatge natural (NLP) centrada en la comprensió d'aquest.
NER	Eina que té com objectiu reconèixer entitats anomenades, per exemple, noms o localitats.
Web Scraping	Tècnica que, mitjançant software, extreu informació de llocs web.

Glossary

Virtual assistant	Software agent that helps users automatizing and doing tasks thanks to human-machine interaction.
Machine Learning	Artificial intelligence subtopic whose objective is to develop techniques that allow machines to learn.
NLP:	Artificial intelligence subtopic that studies the interaction between human language and machines.
NLU	Natural language processing (NLP) subtopic that aims to understand language.
NER	Tool that aims to recognize named entities, for instance, names or locations.
Web Scraping	Technique that uses software to retrieve information from websites.

Agradecimientos

En primer lugar me gustaría dar las gracias al director del proyecto, Javier Béjar. Su disponibilidad, consejos y ayuda en la investigación han sido esenciales para el desarrollo del proyecto.

Además, me gustaría también agradecer a mis compañeros Daniel, Joel y Carla por su apoyo y ánimo durante el desarrollo, y por su ayuda probando y dando *feedback* acerca del funcionamiento del bot a lo largo de todo el desarrollo.

Finalmente me gustaría agradecer a mi familia el apoyo que me han dado no sólo durante el desarrollo del proyecto pero también durante todos los estudios del grado.

Índice de contenidos

Índice de Tablas

Índice de Figuras

Capítulo 1

Introducción

En la actualidad se vive en un momento tecnológicamente centrado en los datos. No solamente se quiere obtener o disponer de datos (como qué tiempo hará mañana, cómo ha terminado un partido de fútbol, dónde está el hospital más cercano, etc), sino que éstos deben ser obtenidos de forma rápida y fácil. Cada vez más se recurre al uso de asistentes personales de los que disponen tanto teléfonos móviles como ordenadores para esa tarea.

El objetivo de este proyecto es desarrollar un asistente en forma de bot conversador, que sea capaz de proporcionar información acerca de la Facultat d'Informàtica de Barcelona (FIB) a los estudiantes de forma rápida y cómoda. Así pues, cualquier estudiante de la FIB podrá preguntarle al asistente dudas y consultas sobre su universidad y obtener la respuesta sin necesidad de entrar en el portal web de la facultad.

Además, se pretende que los estudiantes puedan identificarse con su cuenta del Racó. De esta forma, el asistente va a poder aportar información más concreta de sus estudios y así ejercer de asistente particular al estudiante.

Este documento pretende aportar al lector con una descripción detallada del proyecto. Y por ello está dividido en distintas secciones aportando detalles como el alcance, contexto y planificación (tanto temporal como económica) del proyecto.

Capítulo 2

Contextualización

En esta sección se pretende contextualizar el proyecto y se dividirá en dos apartados. El primero pretende describir cual es el contexto del proyecto explicando la área de interés, y posteriormente definiendo cuáles son los actores implicados. El segundo apartado explicará el estado del arte de las áreas en las que se centra el proyecto.

2.1. Contexto

En esta sección se hará una descripción de cuáles son las áreas de interés de las que trata el proyecto, y posteriormente se definirán cuáles son los actores implicados en el desarrollo del proyecto.

2.1.1. Áreas de interés

Las áreas de interés del proyecto se podrían clasificar en aspectos no técnicos, que hacen referencia a los datos y a la interacción, y a los aspectos técnicos, que incluyen los campos de NLP, el concepto de web scraping.

Datos

Puesto que el proyecto está orientado a poder aportar datos a los usuarios, para poder llevarlo a cabo hace falta saber qué datos se pueden obtener.

En el contexto de la facultad, muchos de los datos están disponibles en la API del Racó [1]. Ésta distingue dos tipos de datos, públicos y privados.

Los primeros hacen referencia a datos de la facultad, tales como qué asignaturas se imparten (conteniendo información como idiomas de impartición, guía docente, etc), plazas disponibles para matricularse en las asignaturas, los distintos departamentos y especialidades (con información como qué competencias tienen, qué asignaturas disponibles hay, quién es el responsable, etc). También contiene noticias, e información relacionada con las reservas de aulas.

Toda esta información es pública y no requiere de ningún tipo de privilegios para ser obtenida.

Los datos privados, requieren estar identificado con la cuenta del Racó para ser obtenidos, y son datos personales sobre el estudiante identificado. Éstos contienen información personal del usuario como nombre, apellidos, correo electrónico de la facultad. También información como las asignaturas que está cursando, los avisos que han publicado profesores para éstas, e incluso su horario.

Además, la información referente a la docencia de la facultad está recogida en el Directori de la UPC [2]. Ahí se puede encontrar para cada docente, su correo electrónico, su despacho (si dispone de él), y cuál es el teléfono de su despacho.

Interacción

Un concepto especialmente importante en este proyecto es el de la interacción. Tratándose de un bot conversacional es importante que la interacción con las personas sea lo más auténtica posible para asegurar una buena experiencia de usuario.

Hay muchas formas en las que un chatbot puede interactuar con los usuarios. La más básica es mediante el mensaje, que permite no solamente mandar texto al usuario, sino que puede hacer referencia o responder a un mensaje en un tiempo anterior (también llamado mención), o también la inclusión de elementos multimedia al mensaje, tales como imágenes, vídeos, audio, etc.

También, una forma en la que se puede interactuar con el usuario es mediante acciones. Las acciones permiten informar al usuario de que el bot está llevando a cabo algún proceso. Éstas definen un "estado", y son entre otras: escribiendo, enviando foto, grabando vídeo, grabando audio, etc. No todas las plataformas permiten definir este tipo de interacción.

Otra forma importante de interacción con un bot suele ser el uso de comandos. Los comandos se mandan como mensaje pero están definidos siguiendo algún patrón, por ejemplo "*/activar_notificaciones*" podría ser un comando para activar notificaciones. De nuevo, no todas las plataformas permiten la definición de comandos.

Aspectos técnicos

En lo referente a los aspectos técnicos, se comenzará describiendo el concepto de *Natural Language Processing* (NLP).

Según Gobinda G. Chowdhury (2003, pág. 51) *"Natural Language Processing (NLP) is an area of research and application that explores how computers can be used to understand and manipulate natural language text or speech to do useful things."* [3].

De la definición podemos entender que NLP engloba el conjunto de técnicas que hacen posible que un ordenador sea capaz de procesar y comprender lenguaje natural. Éste es un campo en constante investigación.

El siguiente concepto importante es el de web scraping. Esencialmente, web scraping es una técnica que se basa en usar software para extraer información de sitios web.

Web scraping es posible gracias a mecanismos llamados scrapers, que son los mecanismos encargados de, automáticamente, recolectar información, pueden analizar su contenido, y en caso de ser necesario, presentarlo en algún formato para ser usado posteriormente.

2.1.2. Actores implicados

Vistos los conceptos que definen el proyecto, a continuación se describirá a quién va dirigido el proyecto, quien se beneficiará de los resultados, y quién lo va a usar.

En primer lugar, el proyecto va dirigido a los estudiantes de la FIB. Éstos serán también los que principalmente van a usar el chatbot para obtener la información que necesiten.

Aún así, aunque los estudiantes de la facultad sean los claros beneficiados del desarollo del proyecto, también de sus resultados se beneficiará la facultad, puesto que con el chatbot obtienen una forma directa de interactuar con sus estudiantes.

2.2. Estado del arte

2.2.1. Sistemas de diálogo

Los sistemas de diálogo cada vez reciben más atención. Originalmente éstos funcionaban gracias a sistemas de reglas manualmente generadas, pero actualmente gracias a la investigación en el campo de las redes neuronales recurrentes se están produciendo avances relevantes en los sistemas de diálogo.

Actualmente distintas técnicas de aprendizaje basadas en Machine Learning han mostrado mejores resultados que los sistemas basados en reglas. La mayoría de chatbots que usan Machine Learning se basan en el uso de Aprendizaje Supervisado (*Supervised Learning*).

Ha habido estudios que han probado a crear un modelo de generación de diálogo basado en *Deep Reinforcement Learning* con resultados prometedores, o incluso combinar *Supervised Learning* y *Reinforcement Learning* generando un modelo híbrido de tal forma que el primero puede generar un modelo con un cierto rendimiento base, y el uso de Aprendizaje por Refuerzo puede ayudar al agente a adaptarse a nuevas situaciones [4].

Además, estos últimos años ha habido un incremental interés en conseguir que los chatbots produzcan conversaciones cada vez más indistinguibles de las conversaciones humanas. Para conseguirlo, hay estudios que han combinado *Reinforcement Learning* para la generación de diálogo, y *Adversarial Learning* como metodo para premiar o penalizar los comportamientos o respuestas más "humanas" [5].

A nivel de arquitectura, uno de los mecanismos que están presentando buenos resultados es el uso de *Long Short-Term Memory* (LSTM), que son un tipo de redes neuronales recurrentes capaces (por arquitectura) de aprender y recordar conceptos a largo plazo [6].

Así pues han surgido distintos métodos que pretenden aprovecharse de éste tipo de mecanismos para generar modelos lingüísticos complejos. En esta línea, un estudio probó que construir un modelo *sequence-to-sequence* usando LSTM aplicado a la traducción de inglés a francés, relativamente poco optimizado podía proporcionar un rendimiento superior en distintos *benchmarks* a los comúnmente usados *Statistical Machine Translation* (SMT) [7][8].

Asimismo, en el campo de los sistemas de diálogo, también han surgido proyectos e investigaciones para generar modelos conversacionales usando LSTM. Uno de ellos, presenta con un sistema similar al de la traducción, entrenado sobre un conjunto de diálogos extraídos de subtítulos de películas. El resultado del estudio, muestra la capacidad de establecer una conversación genérica de forma correcta. Asimismo también demuestra que el modelo conversacional puede funcionar correctamente en dominios más específicos [9][10].

Existen también estudios que prueban de construir modelos conversacionales que emulen el comportamiento de individuos concretos (como personajes de series de televisión), con resultados satisfactorios, que demuestran que los modelos son capaces de asimilar ciertos aspectos de la personalidad de los personajes de los que han aprendido [11] [12].

2.2.2. Asistentes virtuales

Los asistentes virtuales han ganado mucha popularidad estos últimos años. A continuación se revisará el estado del arte de los asistentes virtuales en entornos tanto académicos, profesionales como asistentes personales.

Asistentes virtuales en entornos académicos

En el contexto académico, existe un proyecto que pretende implementar un asistente virtual que permita facilitar la búsqueda de direcciones para los usuarios que se conectan a la web de la facultad de informática de la Universidad Complutense de Madrid [13]. Aunque éste no se trata de un asistente al estudiante como tal, sino pretende facilitar sólo ciertas búsquedas.

Asistentes virtuales en entornos profesionales

A nivel profesional, múltiples son las empresas que han decidido incorporar un asistente virtual a su portal web, puesto que les ofrece reducción en costes y aumento de accesibilidad. La funcionalidad principal suele ser en el apartado de soporte o atención al cliente.

En este contexto existe Nina, un asistente virtual creado por Nuance Communications®, que pretende ofrecer una experiencia de uso muy similar a la de una conversación con un humano, y está orientada a dar soporte a aplicaciones profesionales de múltiples tipos, como por ejemplo, el sector bancario [14].

Asistentes personales

Los asistentes personales son el tipo de asistente personal con más éxito en la actualidad. En esta línea existen gran variedad de asistentes que ofrecen gran variedad de servicios.

El más conocido y mejor valorado es sin duda Google Assistant. Aunque la tecnología que usa no está publicada, da soporte a conversaciones genéricas, también admite preguntas específicas como "¿Qué temperatura hace hoy en Valencia?", respondiendo con el resultado. También, al estar integrado en el sistema operativo de Android, integra la posibilidad de interactuar con aplicaciones del sistema como calendario, reloj, etc [15].

Sistemas similares se pueden encontrar implementados por distintas empresas, como Siri, por Apple [16]. O también por Microsoft, con Cortana [17], o, por último Alexa, de Amazon [18].

2.2.3. Conclusiones

Analizados los resultados de los estudios citados al inicio de esta sección, queda claro que una muy buena solución pasa por usar *Supervised Learning* junto con LSTMs, de forma que no será necesario diseñar ningún mecanismo nuevo.

También hace falta añadir que es interesante el uso de modelos *sequence to sequence* con los propios LSTMs para chatbots de propósito general, pero no necesariamente para un chatbot de propósito específico como el que se presenta en este proyecto.

Capítulo 3

Definición del proyecto

En esta sección se pretende hacer una definición del proyecto empezando por la formulación del problema que se va a estudiar. Seguidamente, se describirá el conjunto de objetivos a tratar y el alcance de los mismos. Posteriormente, se especificará la metodología y pasos a seguir, se terminará hablando de todos los posibles obstáculos y/o riesgos que pueden surgir durante el desarrollo del proyecto.

3.1. Formulación del problema

Actualmente la información acerca de la Facultad de Informática de Barcelona (como qué asignaturas hay, requisitos para cursar alguna asignatura, etc) está distribuida por la página web de la facultad [19], y para una consulta tan simple como saber cuántas plazas libres hay para matricularse en una asignatura requiere ingresar en la web de la facultad (sólo en la versión en catalán), recorrer los menús: *Estudis, Grau en Enginyeria informàtica, Matrícula, Places lliures* para posteriormente buscar entre todas las asignaturas disponibles cuál es la asignatura en cuestión. Éste no es un proceso rápido, ni fácil si no se está familiarizado con la página web de la facultad, o si se trata de un estudiante extranjero que no conoce el catalán.

Asimismo, una parte importante de la rutina de un estudiante de la FIB es estar atento al Racó [20], para poder leer los nuevos avisos de las asignaturas que se están cursando, como publicación de notas, información como cambios de clase, documentación, etc. Ésto requiere acceder a la web, e identificarse con la cuenta del Racó.

La facultad ofrece también un sistema de suscripción que permite recibir correos electrónicos cuando algún profesor de alguna asignatura que se esté cursando publique algún aviso, pero esta suscripción no está activa por defecto, y muchos estudiantes desconocen que existe.

Actualmente se está implementando un panel como sistema de notificaciones de avisos, de forma similar a como funcionan los avisos de Facebook, pero no permitirá que un usuario pueda saber que hay novedades en alguna de las asignaturas que cursa sin necesidad de acceder a la web.

3.2. Objetivos

El principal objetivo del proyecto es desarrollar un asistente a los estudiantes de la facultad, en forma de chatbot. Se pretende desarrollar un bot conversacional que sea capaz de responder a preguntas sobre la facultad de forma cómoda, fácil y rápida.

Asimismo, también se plantea que el asistente sea capaz de asistir a los estudiantes que lo deseen notificando los avisos nuevos que se publiquen en las asignaturas que estén cursando.

Adicionalmente se pretende implementar mecanismos de extracción de información (*information retrieval*), mediante la creación de un scraper que trabajará sobre el Directori de la UPC [2].

Además, el asistente será capaz de comprender y responder a preguntas y consultas en tres idiomas: castellano, catalán e inglés.

La principal fuente de información será la API del racó [1], que contiene información pública acerca de las asignaturas, planes de estudio, etc. Además, ésta ofrece la posibilidad de usar el protocolo Oauth para autentificarse con la cuenta del Racó de los estudiantes para poder acceder a información privada como el horario, los avisos, etc.

3.3. Alcance

Para poder llevar a cabo el proyecto, la infraestructura elegida para desarrollar el asistente es Telegram Messenger, desarrollado por Telegram Messenger LLP, que es una aplicación de mensajería multiplataforma (disponible para Android, iOS, Windows y en forma de cliente web) debido a su popularidad y a la capacidad de ser accedido desde una gran cantidad de dispositivos y plataformas. Telegram Messenger, además, dispone de muchas facilidades para llevar a cabo tal tarea, gracias a la posibilidad de crear bots fácilmente usando su *Application Programming Interface* (API) basada en *Hypertext Transfer Protocol* (HTTP) [21][22].

Aunque la plataforma de Telegram Messenger ofrece una gran cantidad de formas de interacción, el bot se centrará en aprovecharse de los mensajes, los comandos y las acciones (ver sección 2.1 para más detalles).

Puesto a que la API con la cual se trabajará funciona con peticiones HTTP, hay una gran variedad de lenguajes de programación que nos permitirían implementar el bot, no obstante se ha escogido Python (en su versión 3.6.3) para el desarrollo íntegro del proyecto.

La elección de Python está basada en primer lugar por el conocimiento del lenguaje, y en segundo lugar porque es un lenguaje muy maduro, y con gran cantidad de librerías enfocadas a construir modelos de Machine Learning, que serán necesarios para el desarrollo del proyecto, y con una comunidad extensa y muy activa.

Puesto que el asistente funcionará usando lenguaje natural, y además es un asistente o chatbot de propósito específico, se usará el *framework* de Rasa, que ofrece herramientas open-source para la construcción de chatbots orientados a un objetivo. El *framework* de Rasa está formado por Rasa NLU como motor de comprensión del lenguaje natural, Rasa Core como motor de diálogo, y usa distintas técnicas de Machine Learning.

De éstos, se aprovechará las interfícies que proporcionan para implementación sobre Python, que son el intérprete de NLU para detectar intenciones y entidades en los mensajes, y los modelos que ofrece Rasa Core que nos permitirán definir qué posibles acciones debe haber como respuesta por cada intención.

En cuanto al alcance de funcionalidad del bot, es decir, al alcance de información que el bot será capaz de proporcionar, por una parte, a los estudiantes que no se identifiquen con su cuenta del racó podrá responder a preguntas acerca de plazas libres, asignaturas y profesores, es decir, información general de la facultad.

Además, por la parte más privada, el bot será capaz de no sólo responder a preguntas acerca de horarios, prácticas a presentar o exámenes en curso, sino que también podrá proporcionar avisos sobre publicaciones de los profesores sin necesidad de ninguna interacción del usuario.

3.4. Obstáculos y riesgos

A continuación se expondrán cuáles son los principales obstáculos o riesgos que tiene el proyecto.

Se empezará hablando de los obstáculos. Por un lado, uno de los principales obstáculos del proyecto es encontrar soporte en forma de librerías o aplicaciones *Open-Source*e que permitan llevar a cabo tareas como el procesado del lenguaje natural, o traducción para permitir que el chatbot funcione en múltiples lenguajes.

Otro posible obstáculo a tener en cuenta es el tiempo. Al tratarse de un proyecto que usará técnicas de Machine Learning, se debe tener en cuenta el tiempo que lleva generar un modelo. Un tiempo muy extenso podría causar retrasos respecto a la planificación.

En lo referente a los riesgos, el proyecto también conlleva ciertos riesgos. Tal y como se ha descrito en la sección 3.3, al tratarse de un bot de Telegram Messenger, que el asistente funcione correctamente o no, depende del funcionamiento de los servidores de Telegram Messenger. Que éstos dejen de funcionar, o funcionen con retraso, provocará que el funcionamiento normal del asistente sea alterado.

De la misma forma, el proyecto, tal como se ha descrito en 3.2, usa la API del Racó como fuente de información. Como en el caso anterior, un incorrecto funcionamiento del servidor encargado de la API provocará un incorrecto funcionamiento también del asistente.

3.5. Metodología y rigor

La metodología que se va a seguir durante el desarrollo del proyecto será una metodología ágil. Ésta, proporcionará flexibilidad, y permitirá la generación de pequeños prototipos cada vez más funcionales del asistente.

Inicialmente, hasta tener una base estructural suficiente sobre la cual implementar funcionalidades, las iteraciones serán acotadas de forma temporal, semanalmente. Al acabar la base estructural, para validar éstas iteraciones, se revisará de forma exhaustiva el resultado y se verificará que sigue los patrones marcados por el diseño previo.

Posteriormente, una vez la estructura principal haya sido implementada, las iteraciones pasarán a ser basadas en funcionalidades, pretendiendo cerrar una iteración cuando se haya implementado una nueva funcionalidad.

Por último, el método de validación principal para esta segunda fase, será comprobar el grado de satisfacción de uso (por cada funcionalidad), con estudiantes de la facultad. Esto significa que después de implementar una funcionalidad, distintos estudiantes de la facultad la probarán, y del feedback que éstos aporten se continuará en la iteración hasta que haya un alto grado de satisfacción.

Capítulo 4

Planificación temporal

En esta sección se describen cuáles son las tareas a realizadas durante el proyecto, cuáles son los recursos necesarios, y posteriormente se describe la planificación temporal seguida para llevar a cabo cada una de las tareas descritas.

Más tarde se valoran posibles desviaciones o problemas y se definirá cual será el plan de acción. Por último se explican las modificaciones que se hayan llevado a cabo respecto a la planificación original del proyecto.

4.1. Descripción de las tareas

Esta sección pretende definir cada una de las tareas necesarias para llevar a cabo el proyecto, e indicar qué tipo de recursos fueron necesarios para completarlo con éxito. Posteriormente, se especificarán los requisitos tanto *hardware* como software.

4.1.1. Elección de plataforma, lenguaje y *framework*

Esta tarea empezó antes del inicio oficial del proyecto, en el momento en el que se inscribió el trabajo. Gracias a referencias aportadas por el director del proyecto, se investigó acerca de los chatbots, tecnologías existentes, y se revisó cuáles eran las posibles plataformas dónde podría desarrollarse el bot. Una vez decidido que la plataforma sería Telegram Messenger, y que el lenguaje sería Python, se buscó si existían *frameworks* open-source que pudieran facilitar la gestión del diálogo, y finalmente se escogió Rasa [23].

Esta tarea llevó aproximadamente un mes.

4.1.2. Diseño y desarrollo del primer prototipo

Esta tarea consiste en pensar cómo diseñar de forma modular el bot para poder fácilmente alterar, añadir o eliminar funcionalidades. Requiere de haber definido la infraestructura o plataforma y el lenguaje de programación previamente.

Una vez el diseño está hecho, se implementa la base del bot (acorde al diseño), para poder, posteriormente, desarrollar el resto de funcionalidades de forma cómoda y eficiente.

Esta tarea se desarrolló a lo largo de tres semanas y se realizó en paralelo con el módulo de GEP.

4.1.3. Desarrollo del módulo de GEP

Esta tarea consiste en la redacción de los distintos documentos que requiere el módulo de GEP, así como de la preparación de la presentación final presencial. Se llevó a cabo durante los primeros dos meses del curso lectivo, en paralelo al desarrollo del proyecto.

4.1.4. Implementación de funcionalidades básicas

En esta tarea se diseña e implementa las funcionalidades básicas y necesarias para el bot. Ello requiere que la estructura base y el diseño de los módulos esté definida, tal como lo indica el paso anterior.

En esta fase se definen e implementan funcionalidades como el acceso a la API del Racó (tanto de forma pública como privada, implementando la autenticación por el protocolo Oauth), la gestión de los comandos como "/start", "/login", etc, para interactuar de forma básica con el bot, un módulo que se encargue de gestionar los mensajes, y un módulo que se encargue de registrar información necesaria de los usuarios con los que el bot interactúe.

Esta tarea llevó alrededor de un mes y coexistió con el desarrollo del módulo de GEP.

4.1.5. Creación del *scraper* y *multithreading*

Esta tarea consiste en implementar el *scraper* que permita recoger, dar formato y almacenar la información del Directori UPC. La creación del *scraper* en si no depende de ninguna tarea anterior.

También, se diseñará e implementará la capacidad de disponer de múltiples *threads* para, por ejemplo, poder realizar tareas como notificar a los usuarios de los avisos que se publiquen, de forma paralela al funcionamiento normal del bot (atendiendo a preguntas que se le puedan hacer). Ésta parte de la tarea requiere de módulos como el módulo de gestión de mensajes, y el que registra información de los usuarios, por lo que debe llevarse a cabo posteriormente a las funcionalidades básicas.

Esta tarea llevó 2 semanas.

4.1.6. Integración de Rasa al proyecto

Con todas las tareas anteriores hechas, queda la parte más importante del proyecto, que es la integración de Rasa (tanto de Rasa NLU como de Rasa Core) al proyecto. Esta tarea requiere que las funcionalidades básicas (como el gestor de mensajes) estén implementados. Asimismo también depende, en parte, de la existencia de los datos de profesores, por lo que debe hacerse después de la creación del *scraper*.

En esta tarea se diseña y genera un *dataset* de preguntas acerca de la facultad (asignaturas, profesores, etc), y también se entrena modelos de conversación para poder alimentar el *framework* de Rasa. Posteriormente, de forma independiente al bot se busca la construcción de un modelo fiable del NLU y del sistema de diálogo. Posteriormente, éste se integrara a la arquitectura existente del bot.

Este proceso llevó unas 5 semanas y coexistió con la redacción de la memoria.

4.1.7. Confección de la memoria

Esta tarea consiste en la redacción de la memoria final a partir del documento de GEP. Empezó unas tres semanas antes del hito intermedio y se prolongó hasta la entrega de éste a finales de junio, por lo que tuvo una duración aproximada de un mes y medio, y se llevó a cabo paralelamente al desarrollo del proyecto.

4.1.8. Implementación de las acciones y testeo final

Esta tarea consiste en definir qué acciones puede llevar a cabo el bot, es decir, cuando detecta que se le está solicitando un dato (mediante el uso del *framework* Rasa, que se implementó en el paso anterior), qué debe hacer para proporcionarlo. Así pues, requiere que la integración con Rasa esté ya desarrollada.

Asimismo, también una vez las acciones estén implementadas, es necesario destinar mucho tiempo al testeo pasa asegurarse de un correcto funcionamiento.

Esta tarea duró alrededor de un mes, y se llevó a cabo junto con la preparación de la defensa.

4.1.9. Preparación de la defensa

Durante esta tarea se prepara la defensa del proyecto, que consistirá en preparar las transparencias de apoyo y el ensayo. Empezó unas tres semanas antes de la fecha señalada para la defensa hasta la presentación final, por lo que su duración es de 3 semanas, y se llevó a cabo junto con el resto de tareas. Ésta tan solo requiere que parte del testeo se haya llevado a cabo.

Por último, a continuación, la tabla 1 muestra las dependencias entre las fases comentadas.

Fase	**Título**	1	2	3	4	5	6	7	8	9
1	Elección de plataforma, lenguaje y *framework*									
2	Diseño y desarrollo del primer prototipo	X								
3	Desarrollo del módulo de GEP	X								
4	Implementación de funcionalidades básicas		X							
5	Creación del *scraper* y *multithreading*				X					
6	Integración de Rasa al proyecto				X	X				
7	Confección de la memoria			X						
8	Implementación de las acciones y testeo final					X	X			
9	Preparación de la defensa								X	

Tabla 1: Tabla de dependencias entre fases del desarrollo.

4.2. Recursos usados

A continuación se describirán cuáles son los recursos tanto *hardware* como software para el desarrollo del proyecto.

4.2.1. Recursos *hardware*

- Ordenador de sobremesa, que será usado principalmente para entrenar modelos en la tarea de integración de Rasa al proyecto. Tiene las siguientes especificaciones : Intel Core i5 4400 a 3.1GHz, 16GB RAM DDR3, nVidia GeForce GTX 1060.
- Ordenador portátil Acer Aspire V3-572G, que será usado para la implementación del proyecto y para redactar la documentación. Tiene las siguientes especificaciones:
  Intel Core i7 5500U a 2.4GHz, 8GB RAM DDR3, nVidia GeForce GT840M.
- Teléfono móvil Xiaomi Redmi Note 4, para poder testear (a lo largo del desarrollo pero muy profundamente en la última tarea) mediante Telegram Messenger en plataforma portable. Tiene las siguientes especificaciones:
  Qualcomm Snapdragon 625 *Octa-Core* a 2GHz, 4GB RAM.

4.2.2. Recursos software

- Windows 10, como único sistema operativo para ordenadores.
- Android 7.0 Nougat, como sistema operativo portable.
- Anaconda, paquete de software orientado a *Machine Learning* para Python (usado en la integración de Rasa).
- Rasa Core y NLU, librerías orientadas a la construcción de bots conversacionales (usado en la integración de Rasa).

- Atom y Github, como editor de código y repositorio para el código a lo largo de todo el proyecto.
- Telegram Messenger, como plataforma de mensajería usada en el proyecto.
- LaTeX, para toda la documentación del proyecto.

4.2.3. Recursos humanos

Todas las tareas requieren recursos humanos.

- Elección de plataforma, lenguaje y framework: Investigación.
- Diseño y desarrollo del primer prototipo: Diseño e implementación.
- Desarrollo del módulo de GEP: Redacción.
- Implementación de funcionalidades básicas: Implementación.
- Creación del scraper y multithreading: Diseño de los threads e implementación.
- Integración de Rasa al proyecto: Investigación de la librería, e implementación.
- Confección de la memoria: Redacción.
- Implementación de las acciones y testeo final: Implementación y testeo.
- Preparación de la defensa: Preparación de transparencias, ensayo.

4.3. Planificación temporal

4.3.1. Distribución del tiempo por tareas

En la tabla 2 se puede observar la cantidad de horas que se han dedicado a cada tarea.

Tarea	**Duración estimada (h)**
Elección de plataforma, lenguaje y *framework*	70
Diseño y desarrollo del primer prototipo	55
Desarrollo del módulo de GEP	60
Implementación de funcionalidades básicas	60
Creación del *scraper* y *multithreading*	50
Integración de Rasa al proyecto	75
Confección de la memoria	60
Implementación de las acciones y testeo final	85
Preparación de la defensa	35
Total	**550**

Tabla 2: Tiempo dedicado para el desarrollo de las tareas.

4.3.2. Diagrama de Gantt

En el Anexo, la figura 26 muestra el diagrama de Gantt (generado con `https://www.teamgantt.com`) del desarrollo del proyecto.

4.4. Evaluación de alternativas y plan de acción

Si bien la planificación original deja suficiente tiempo para poder llevar a cabo el proyecto en los términos temporales definidos, es posible que surjan imprevistos en el desarrollo, y para poder llevar a cabo el proyecto hace falta definir un plan de acción.

La principal prioridad del proyecto es tener un asistente virtual que funcione vía Telegram Messenger y sea capaz de contestar a las preguntas generales acerca de la facultad. Así pues en caso de que haya retrasos o problemas, se asignará más tiempo a llevar a cabo esta tarea, quitándolo de tareas como creación del scraper o multithreading (puesto que el asistente no requeriría de estos elementos en tal caso).

A partir de aquí, que el bot sea capaz de reconocer y responder a más o menos tipos de preguntas es otro elemento fácil de ajustar al ritmo de desarrollo del bot y al tiempo disponible. De tal manera que en caso de que alguna funcionalidad requiera más tiempo del planificado, se puede compensar, reduciendo en mayor o menos mesura la capacidad del bot de responder a tipos distintos de preguntas (priorizando siempre las preguntas que se consideren más importantes y/o útiles).

Otro posible desvío sería que el tiempo que tardase el ordenador en entrenar el modelo de diálogo o de NLU (en la integración con Rasa) fuera muy largo, y eso provoque retrasos.

En tal caso se intentará paralelizar al máximo el desarrollo y avanzar o hacer una primera implementación de elementos posteriores como algunas acciones, o avanzar en cuanto a documentación mientras se entrena el modelo.

Así pues, en caso de que hubieran desviaciones respecto a la planificación descrita, ni la duración ni el consumo de recursos del proyecto variarían, pero sería posible que si que lo hiciera el resultado final del proyecto a nivel de funcionalidades, como se ha comentado.

4.5. Desviaciones en la planificación

Aunque se ha definido un plan de acción en caso de desviaciones, no ha sido necesario llevarlo a cabo puesto a que la planificación original para el proyecto se ha respetado y se ha podido completar el desarrollo tal como se había planificado originalmente.

Capítulo 5

Presupuesto y sostenibilidad

5.1. Presupuesto

Para desarrollar el proyecto con éxito, han sido necesarios los recursos mencionados en la sección anterior. En esta sección se presenta el coste del proyecto, teniendo en cuenta tanto los costes en recursos humanos, *Hardware* y *Software*, y además las correspondientes amortizaciones. También se considerarán también los costes indirectos del proyecto.

Para calcular las amortizaciones, se ha usado la siguiente fórmula:

$$amortizacion = \frac{550\ horas\ proyecto}{años\ vida\ util \cdot 220\ dias\ laborables \cdot 8\ horas/dia} \cdot precio\ producto$$

5.1.1. Presupuesto en Recursos Humanos

Puesto que el proyecto se ha desarrollado íntegramente por una sola persona, ésta ha realizado los 4 roles necesarios para el correcto desarrollo, que son las de cabeza de proyecto, diseñador de software, programador y *tester*. La tabla 3 muestra los costes en recursos humanos (los precios por hora se han aproximado a partir de ofertas publicadas en `https://www.pagepersonnel.es`):

Rol	**Horas**	**€/hora**	**Coste**
Cabeza de proyecto	130	50	6.500€
Diseñador de *Software*	120	35	4.200€
Programador	215	30	6.450€
Tester	85	20	1.700€
Total	**550**		**18.850€**

Tabla 3: Presupuesto en recursos humanos.

A continuación, la tabla 4 muestra la distribución del tiempo que cada rol ha invertido en cada etapa, siguiendo el orden descrito en la planificación del proyecto.

Etapa	**Duración**	**Cabeza proyecto**	**Diseñador *Software***	**Programador**	***Tester***
1	70h	40h	20h	10h	0h
2	55h	0h	35h	20h	0h
3	60h	40h	10h	10h	0h
4	60h	0h	10h	35h	15h
5	50h	0h	10h	30h	10h
6	75h	0h	15h	60h	0h
7	60h	10h	20h	20h	10h
8	85h	5h	0h	30h	50h
9	35h	35h	0h	0h	0h
Total	**550h**	**130h**	**120h**	**215h**	**85h**

Tabla 4: Distribución de horas por rol.

5.1.2. Presupuesto en *Hardware*

Para poder llevar a cabo el proyecto, se ha usado el *Hardware* descrito en la planificación del proyecto.
La tabla 5 resume los costes en *Hardware* del proyecto.

Producto	**Precio**	**Unidades**	**Vida útil**	**Amortización**
Ordenador de sobremesa	1000€	1	4 años	78€
Acer Aspire V3-572G	600€	1	4 años	47€
Xiaomi Redmi Note 4	200€	1	3 años	21€
Total	**1.800€**			**146€**

Tabla 5: Presupuesto en *Hardware*.

5.1.3. Presupuesto en *Software*

La tabla 6 muestra los costes en *Software* del desarrollo del proyecto.

Producto	**Precio**	**Unidades**	**Vida útil**	**Amortización por unidad**
Windows 10	130€	2	3 años	13.5€
Android 7.0 Nougat	0€	1	3 años	0€
Anaconda	0€	2	3 años	0€
Rasa Core y NLU	0€	2	3 años	0€
Atom y Github	0€	2	3 años	0€
Telegram Messenger	0€	1	3 años	0€
LaTeX	0€	1	3 años	0€
Total	**260€**			**27€**

Tabla 6: Presupuesto en *Software*.

5.1.4. Costes indirectos

La tabla 7 muestra los costes indirectos, que no son considerados en ninguna de las categorías anteriores.

Producto	**Precio**	**Unidades**	**Precio estimado**
Electricidad	0,12€/kWh	250 kWh	30€
Acceso a internet	30€/mes	5 meses	150€
Total			**180€**

Tabla 7: Costes indirectos.

5.1.5. Costes imprevistos

Para poder disponer de cierto margen para imprevistos, se planificó una parte de los costes para tales. La partida para imprevistos fue de un 10 % sobre la suma de costes directos e indirectos.

Aún así, durante el desarrollo del proyecto no ha surgido ningún imprevisto, por lo que no ha sido necesario usarlo.

5.1.6. Plan de contingencia

Como se ha comentado en la planificación, existía el riesgo de que alguna de las fases terminara durando más de lo planificado, especialmente la fase de integración de rasa que requiería de esperas para entrenar modelos, y la fase final de testeo.

Se estimó un riesgo de un 40 % de que las fases comentadas duren más de lo esperado, y el coste podría suponer un incremento de horas siguiendo la tabla 8:

Rol	**Horas**	**€/hora**	**Coste**
Cabeza de proyecto	10	50	500€
Diseñador de *Software*	20	35	700€
Programador	30	30	900€
Tester	20	20	400€
Total	**80**		**2.500€**

Tabla 8: Riesgo por incremento de horas.

Así pues, la partida de contingencia que se preparó fue de 1.000€. Aún así, finalmente no ha sido necesario usarla debido a que ninguna de las fases se ha retrasado.

5.1.7. Presupuesto total

Teniendo en cuenta todos los factores mencionados anteriormente, a continuación, en la tabla 9 de detallará el presupuesto total del proyecto.

Concepto	**Coste Estimado**
Recursos Humanos	18.850€
Recursos *hardware*	146€
Recursos Software	27€
Costes indirectos	180€
Total	**19.203€**

Tabla 9: Presupuesto total.

5.2. Control del presupuesto

Como a lo largo del documento se ha ido especificando, pueden haber modificaciones principalmente a nivel temporal respecto a la planificación descrita.

Así pues, es muy poco probable que se requiera de más recursos *hardware* que los ya citados anteriormente en las estimaciones. De forma similar, es bastante poco probable que se requiera de más recursos software, aunque en caso de que fuera necesario, muy probablemente se podrían encontrar los recursos necesarios de forma open-source y así no provocar ningún cambio en el presupuesto planteado.

En caso de no existir alternativas open-source, hay planificadas económicas partidas para casos como éstos.

Así pues, el coste que realmente hace falta controlar es el relacionado con los recursos humanos necesarios. En caso de que el proyecto requiera más horas de trabajo por alguno o algunos de los roles, hay una partida para contingencias planificada que debería cubrir ese coste, aunque sería necesario controlarlo por si con ella no hubiera suficiente.

Así pues, al finalizar cada tarea, se controlará el coste de los recursos humanos, y se comparará con las estimaciones hechas. Así, se podrá prever si el desarrollo del proyecto conllevará costes extras superiores a los planificados.

Para hacerlo mediremos las desviaciones en el coste de realización de tareas de la siguiente forma:

$$Desviacion = (coste\ estimado - coste\ real) * consumo\ horas\ reales$$

Dónde el coste estimado y el real hace referencia al coste en recursos humanos.

5.3. Sostenibilidad

En esta seccion se evaluará la sostenibilidad del proyecto en tres áreas distintas: la área económica, la social y la medioambiental.

5.3.1. Resumen de la evaluación de sostenibilidad

Personalmente creo que la universidad me ha hecho entender la importancia del desarrollo sostenible, y en este sentido me ha hecho ser consciente del problema. Asimismo, creo que el aprendizaje acerca de éste que se ha hecho es insuficiente. Muy pocas veces he tenido la ocasión de valorar la sostenibilidad (sea económica, social o ambiental) en algun proyecto a lo largo del grado, por lo que considero que tengo una capacidad reducida para valorar o aplicar de forma correcta métodos o métricas para valorar el impacto de los proyectos que desarrollo, o para intentar reducirlo.

A nivel del código deontológico, gracias a varias asignaturas optativas que he realizado, he tenido la oportunidad de leer y entender los códigos de comportamiento que definen un ingeniero informático, y por eso considero que en este aspecto a nivel profesional estoy perfectamente capacitado.

Finalmente, a nivel colaborativo considero que la facultad si que ha tenido un impacto positivo importante, y me ha proporcionado la información necesarias para poder aplicar herramientas de trabajo colaborativo a los proyectos y poder ver el impacto y la eficiencia que tienen en un proyecto.

Como conclusión, considero que aunque la facultad ha proporcionado todas las herramientas necesarias, y ha conseguido que sea consciente del problema que hay actualmente, no creo que me haya preparado para poder medir el impacto que tienen los proyectos que desarrollo, o aplicar medidas sostenibles a éstos.

5.3.2. Sostenibilidad económica

Este documento describe los costes del proyecto teniendo en cuenta múltiples factores.

La solución propuesta es difícil de abaratar de forma contundente. Se podría prescindir de usar sistema operativo Windows para el desarrollo, o de reducir el coste del *hardware* (usando tan solo el ordenador portátil), pero eso reduciría muy ligeramente los costes *hardware* y *software* (hasta 140€ de ahorro), y aumentaría otros factores como el tiempo de cómputo en una cantidad notable de horas, que muy probablemente conllevarían un coste muy superior al ahorro.

Aumentar el tiempo de desarrollo no es deseable, debido a que el principal coste es en Recursos humanos, y la complejidad del proyecto requiere ya de una gran cantidad de horas en desarrollo y diseño.

Rebajar la complejidad (implicando menos tiempo de desarollo) para abaratar el coste significaría reducir la funcionalidad del bot, lo cual lo haría menos atractivo e útil, y consecuentemente generaría menor interés a las partes interesadas (es decir, la universidad y los estudiantes).

Así pues, el principal problema a nivel económico del proyecto es la dificultad de monetizarlo. Al no existir proyectos similares es difícil definir un precio que sea razonable y permita generar beneficios a medio y largo plazo.

Existen distintas posibilidades. La primera y más viable es que la facultad pague de forma periódica por su implantación y mantenimiento. Otra alternativa es la de, extender el proyecto de forma futura, y definir tipos de usuarios (gratuito y de pago), y ofrecer ciertas funcionalidades como el servicio de notificaciones, sólo a los usuarios que paguen por el uso del bot.

5.3.3. Sostenibilidad social

El proyecto de forma personal aportará conocimiento sobre la interacción humana con máquinas, y muchos conceptos poco tratados de forma práctica en la universidad como planteamiento económico o sostenible.

En otra línea, el proyecto pretende mejorar la experiencia de uso y facilitar a posibles de los estudiantes que actualmente tienen que buscar información de forma puntual o periódica en la web de la facultad.

Para hacerlo se escogió una aplicación de mensajería gratuita y multiplataforma, se va a hacer que el chatbot sea capaz de asistir a los estudiantes en los tres principales idiomas hablados en la facultad (catalán, castellano e inglés) para poder ser útil a la máxima cantidad de estudiantes posibles, y el rango de funcionalidades será tan amplio como lo permita el tiempo disponible de desarrollo. Así pues, el desarrollo del proyecto afectará de forma positiva a la comunidad de estudiantes de la facultad.

También es importante analizar qué impacto puede tener el proyecto en la facultad, pues el proyecto puede ayudar a atraer a estudiantes, o conseguir que se fortalezca la relación estudiante-facultad. Además, proporciona un método rápido y muy directo de interactuar con los estudiantes.

Por último cabe decir que es en cierto modo necesario facilitar la búsqueda de información en la facultad. Que esta búsqueda sea más eficiente mediante un bot conversacional o por otros métodos es lo que se evaluará en el proyecto.

5.3.4. Sostenibilidad ambiental

Si bien todos los recursos usados para la realización del proyecto se han descrito en la planificación, a nivel ambiental el proyecto tiene un impacto bastante bajo, y principalmente causado por el consumo eléctrico, que como se ha especificado, no es muy significativo.

Hay ciertas acciones que podrían ayudar a reducir el impacto ambiental del proyecto, como el uso de un solo ordenador (el de sobremesa). El motivo por el que no se ha hecho es por la necesidad de trabajar a distancia en el proyecto y para no extender el tiempo de desarrollo.

También hace falta tener en cuenta que al reusar código como las librerías que ofrece Rasa, se ha ahorrado tiempo y consiguientemente recursos como electricidad, pero no hay ninguna otra parte de la implementación que se pueda evitar reusando código.

Suponiendo que el proyecto estuviera en funcionamiento, se debería de aprovechar algún equipo antiguo, o alguno que actualmente ya esté operativo para poder ejecutar el bot, teniendo en cuenta que los requisitos computacionales del proyecto son mínimos (una vez el modelo se haya generado).

Por último, es difícil valorar qué mejora o aporta el proyecto a nivel ambiental respecto soluciones existentes, porque actualmente no se dispone de ninguna solución similar.

Integración de conocimientos

El proyecto usa elementos de dos grandes disciplinas del campo de la computación, que són *Machine Learning* e *Information Retrieval*.

Por un lado, en el terreno de *Machine Learning*, el proyecto se aprovecha del aprendizaje hecho durante el grado acerca de conceptos y técnicas como los clasificadores, y distintos métodos en los que se pueden llevar a cabo, más concretamente las *Support Vector Machine* (SVM), sus parámetros, y técnicas para optimizarlos (como gridsearchCV). Estos conocimientos se han obtenido en las asignaturas APA y MD.

Por otro lado, en el terreno de *information retrieval*, se han usado técnicas de extracción de información tales como los scrapers, y métodos de procesado de la información como por ejemplo la tokenización de texto. Éstas técnicas se han aprendido en la asignatura de CAIM.

Las técnicas relacionadas con *Machine Learning* serán de utilidad para dar al bot la capacidad de comprender las consultas y saber escoger la respuesta adecuada, mientras que las técnicas de *Information retrieval* serán útiles para recolectar información que no existe en la api del Racó, y a su vez ayudará a procesar los mensajes de entrada.

Identificación de leyes y regulaciones

Una ley o reglamento que podría incidir en el desarrollo del proyecto es el Reglamento General de Protección de Datos (RGPD), que entrará en vigor a nivel europeo el 25 de Mayo de 2018. Esta ley derogará la actual LOPD.

Según la RGPD, se debería valorar proteger los datos en función de varios factores como la sensibilidad de los datos, si éstos configuran un perfil, y provienen de tecnologías invasivas de privacidad (como geolocalización).

En este sentido, el proyecto sólo registra 2 datos que podrían clasificarse como sensibles, que son el token de acceso y de refresco de cada usuario a su información de la API del Racó. Los tokens en si no proporcionan ningún tipo de información sensible ni privada. Además, para poder ser usados con el objetivo de obtener información como el correo electrónico del estudiante, son necesarias 2 claves pertenecientes a la aplicación del bot que se almacenan como variables de entorno, por lo que no figuran en ningún fichero ni registro.

Además de ello, se ha decidido encriptar ambos tokens para cada usuario de forma que en los registros aparezcan encriptados y no puedan ser usados. Así pues, el proyecto respeta el reglamento en éste sentido.

Además, la aplicación no registra ningún tipo de actividad de los usuarios con los que interactúa.

Capítulo 6

APIs, Frameworks y librerías

6.1. API de Telegram Messenger

Telegram Messenger es una aplicación de mensajería instantánea, multiplataforma, que además permite la creación de bots. Para crear uno es suficiente con hablar con BotFather (disponible en `https://telegram.me/BotFather`), un bot creado por Telegram que se encarga de la gestión de los bots.

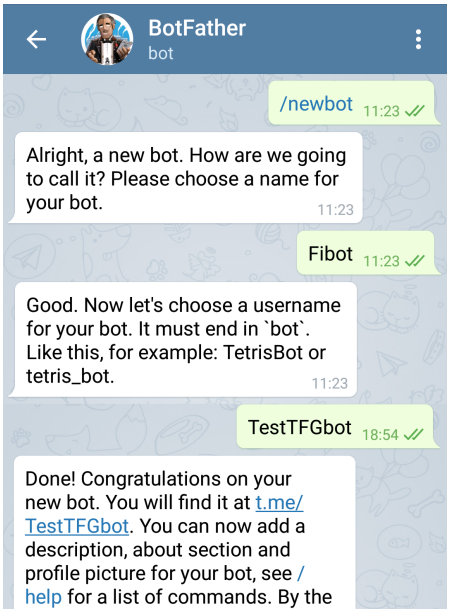

Figura 1: Creación de un bot mediante BotFather.

BotFather permite crear bots, como se puede ver en la figura 1, editar elementos como la descripción, la imagen de perfil del bot, facilitar la lista de comandos para que los usuarios puedan acceder más rapidamente (como se puede ver en la figura 2), y eliminarlos.

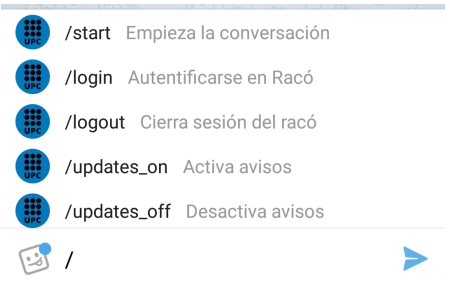

FIGURA 2: Atajo para comandos.

Después de la creación de un bot, BotFather proporciona el token de acceso al bot. Este token es una larga cadena de caracteres que sirve para controlar el bot mediante la API HTTP que Telegram ofrece. Esta API funciona haciendo peticiones HTTP GET a la dirección `https://api.telegram.org/bot<token_acceso>/funcion?parametros`.

La API ofrece una gran variedad de funciones y opciones para los bots, que van desde mirar los mensajes recibidos hasta programar juegos. Pero teniendo en cuenta el alcance del proyecto, se usan principalmente 3 funciones.

En primer lugar, la función **getUpdates** permite recibir toda la información de mensajes y comandos recibidos desde la última revisión de forma estructurada y fácil de interpretar, como muestra la figura 3.

FIGURA 3: Información proporcionada por la función getUpdates.

En segundo lugar, la función **sendMessage** permite mandar mensajes a un destinatario y admite múltiples parámetros que permiten modificar la forma en que se presenta el mensaje, o si el mensaje está respondiendo a otro. Entre todas las opciones que ofrece la funcion, el bot usa:

- **chat_id**: ID del chat del destinatario del mensaje.

- **text**: Texto del mensaje.
- **parse_mode**: Indica si el mensaje será interpretado como texto plano, Markdown o HTML. Sirve para representar elementos en negrita, cursiva o como hiperenlaces.
- **reply_to_message_id**: Permite al bot responder a un mensaje enviado previamente, como se puede ver en la figura 4.

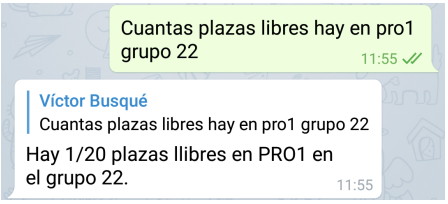

FIGURA 4: Respuesta a un mensaje previo.

Finalmente, la última función que usa el bot es **sendChatAction**, que permite al bot avisar al usuario de que está procesando el mensaje o escribiendo la respuesta. Como parámetro solo toma el chat_id del destinatario de la acción, y qué acción se quiere mandar, que generalmente es "escribiendo". Un ejemplo de la ejecución de esta acción se puede ver en la figura 5

FIGURA 5: Resultado de enviar una acción de "escribiendo".

6.2. Framework RASA

RASA es una empresa con base en Berlín que desarrolla software para ofrecer soluciones conversacionales de forma *Open-Source* en Python.

El hecho de que sea *Open-Source* proporciona muchas ventajas respecto a alternativas como api.ai o wit.ai (de Google y Facebook respectivamente), puesto que los datos potencialmente sensibles no tienen porque pasar por terceros, y al disponer del framework localmente instalado, los chatbots pueden funcionar aunque el software deje de mantenerse.

RASA ofrece dos productos [24]:

- **RASA NLU**: Componente encargado de la comprensión del lenguaje natural. Compatible con los idiomas con corpus disponibles en spaCy (más información en la sección 6.3). Ofrece varios métodos para obtener características de los mensajes y realizar el procesado del mensaje y su posterior transformación a datos estructurados.

- **RASA Core**: Componente de gestión de diálogo totalmente conectado con RASA NLU, que pretende sustituir la máquina de estados clásica de los chatbot tradicionales. El componente puede generar agentes que mediante la interpretación de los mensajes (usando RASA NLU), decidan qué acciones tomar.

El framework se integra con librerías populares del entorno de Python, tales como spaCy o MITIE para el procesado de lenguaje natural, y SKlearn o Keras para los componentes de Machine Learning.

6.3. spaCy

spaCy es una librería muy popular de procesado del lenguaje natural para Python. La librería ha estado diseñada para ser rápida y eficiente, e incluye una gran variedad de funcionalidades.

Las siguientes funcionalidades son las que se aprovechan en el proyecto:

- **Tokenización**: Se usa spaCy para hacer el análisis léxico como parte del procesado de los mensajes antes de su interpretación.
- **Soporte al inglés y castellano**: spaCy dispone de varios corpus para tanto el inglés como el castellano. Para el proyecto se usan los corpus más ligeros para ambos idiomas.
- **Word vectors pre-entrenados**: spaCy dispone de word vectors pre-entrenados para todos los corpus de los que dispone. Éstos son vectores de una dimensionalidad fija que permiten representar de forma numérica, el aspecto semántico de las palabras.

  Los corpus pequeños (como los que se usan para el proyecto) vienen con lo que denominan context-sensitive tensors en lugar de word vectors. Éstos son vectores en 384 dimensiones, que funcionan como los word vectors, pero debido a que han sido calculados sobre un corpus mucho menor, son menos precisos.
- ***Part of Speech tagging***: Es capaz de realizar *Part of Speech tagging* (POS tagging o etiquetado gramatical) para descubrir la categoría gramatical de las palabras.

Además de estos elementos usados en el proyecto, también proporciona otras funcionalidades que resultan interesantes:

- **Named Entity Recognition**: Permite identificar entidades concretas (como nombres de persona, entidades geopolíticas, cantidades, etc) usando un modelo pre-entrenado y muy eficaz.
- **Herramientas de visualización**: Incorpora muchas herramientas que permiten visualizar a nivel sintáctico y también permite ver las *Named entities*.

Capítulo 7

Elementos del bot

En éste capítulo se especificará qué elementos conforman el bot, y se detallará el funcionamiento de cada componente.

7.1. Bases de datos

Para que el bot pueda recordar cierta información acerca de los usuarios con los que habla, o para poder aportar información acerca de profesores, que no está presente en la api del racó, es necesario que disponga de bases de datos.

Para el proyecto se han construido dos bases de datos, una con información acerca de los usuarios, y otra con información acerca de profesores.

7.1.1. Base de datos de usuarios

El bot necesita registrar cierta información acerca de los usuarios, es decir los estudiantes, para poder llevar a cabo varias de sus funcionalidades.

Más concretamente, el bot registrará para cada estudiante con el que interactúe los siguientes datos:

- **Nombre**: Nombre de usuario de la cuenta de Telegram del estudiante. Se usará en ciertos mensajes para humanizar el comportamiento del bot.
- **Idioma**: Idioma con el que se llevará a cabo toda la comunicación con el bot. Puede ser 'en' para el inglés, 'es' para el castellano, o 'ca' para el catalán.
- **Token de acceso**: Si el estudiante se ha identificado con la cuenta del Racó no contendrá ningún dato. Sino, tendrá una cadena de 44 caracteres resultado de encriptar el token de acceso del estudiante usando el *Advanced Encription Standard* (AES) y codificar el resultado en base64. El token que se encripta permite acceder a la información particular del estudiante como el horario, los avisos, prácticas, etc, y caduca a las 10 horas de ser proporcionado.

- **Token de refresco**: De la misma forma que el dato anterior, en caso de que el estudiante esté identificado con su cuenta del racó, se almacenará el valor encriptado y codificado (tal como se hace con el token de acceso) del token de refresco. Este token sirve para obtener un nuevo token de acceso a partir del anterior en caso de que haya caducado.
- **Estado actual**: Este dato almacena en qué estado está el usuario. Actualmente tan solo existen dos estados, pero de esta forma en caso de que sea necesario extender la cantidad de estados el bot tiene soporte para ello. El dato tiene valor 0 si el bot está esperando una pregunta o consulta del usuario, y en valor 1 si el bot está esperando a que el usuario proporcione el código de autenticación cuando se identifica en el Racó (más sobre esto en la sección 7.2.1).
- **Expiración del token**: Este campo almacena de forma estructurada cuando el token de acceso a los datos caducará. Contiene información acerca del día, mes, año, hora, minuto y segundo en el que caducará el token de acceso.
- **Identificación**: Dato que indica si el usuario está actualmente identificado con su cuenta del Racó o no.
- **Notificaciones**: Dato que indica si el usuario quiere recibir mensajes de forma automática con los avisos nuevos que tenga en el Racó o no.

Toda esta información se almacenará en un fichero json, indexado por el chat id, que es un valor numérico único que identifica a una conversación con un usuario en Telegram.

7.1.2. Base de datos de profesores

Como se ha comentado, la api del Racó contiene información muy escasa acerca de profesores, por lo que ha sido necesario obtenerla de otras fuentes. Para este proyecto, la fuente ha sido el Directori de la UPC, de dónde se ha extraído la información mediante un scraper (más información en la sección 7.9.1).

Esta base de datos se compone por 10 ficheros json, cada uno con información de los profesores que forman parte de los distintos departamentos que forman la facultad, que son:

- **AC**: Departamento de Arquitectura de Computadores.
- **CS**: Departamento de Ciencias de la Computación.
- **EIO**: Departamento de Estadística e Investigación Operativa.
- **ESAII**: Departamento de Ingeniería de Sistemas, Automática e Informática Industrial.
- **ESSI**: Departamento de Ingeniería de Servicios y Sistemas de Información.
- **FIS**: Departamento de Física.

- **MAT**: Departamento de Matemáticas.
- **OE**: Departamento de Organización de Empresas.
- **THATC**: Departamento de Teoría e Historia de la Arquitectura y Técnicas de Comunicación.
- **IRI**: Departamento del Institut de Robòtica i Informàtica industrial.

Para cada profesor de los departamentos comentados, se almacenarán dos datos, indexados mediante el nombre completo del profesor. Estos datos son:

- **Correo electrónico**: La dirección de correo electrónico de la UPC. Si no se dispone de él, el dato quedará vacío.
- **Dirección del despacho**: Para los profesores con despacho en el Campus Nord se almacena la dirección dónde se ubica su despacho. Si no se dispone de él se almacenará como vacío.

Como en el medio en el que se usará el bot es una aplicación de mensajería instantánea, es importante prever que la forma en la que se reciban los nombres de profesores no necesariamente coincida con el nombre almacenado en la base de datos. Para poder encontrar cuál es el profesor al que se está haciendo referencia, se buscará el profesor de la base de datos que tenga el nombre con la distancia de edición (*edit distance* o distancia de Levenshtein) menor respecto al nombre introducido.

7.2. Mecanismos de interacción con la API del Racó

La principal fuente de información para el bot será la API del Racó. Ésta divide los datos que puede proporcionar en dos categorías.

Por un lado los datos públicos, que son los datos acerca de la facultad, tales como planes de estudio, información sobre asignaturas como la guía docente, información acerca de exámenes, noticias, etc.

Por otro lado, también hay datos privados para los que se requiere de un token de acceso de un estudiante obtenido mediante la autenticación Oauth 2.0 con la API del Racó. Estos datos contienen información acerca de los estudios particulares del estudiante como el horario, los avisos de las asignaturas que se están cursando, las prácticas abiertas de las asignaturas, etc.

Por ese motivo, es necesario que el bot tenga mecanismos para poder obtener esta información, y para hacerlo se han desarrollado dos elementos. El primero es para llevar a cabo la comunicación necesaria del protocolo Oauth 2.0 que requiere la API, y el segundo para poder obtener la información tanto pública como privada.

7.2.1. Oauth

Como se ha comentado, la API del Racó usa el protocolo Oauth en su versión 2.0 para dar acceso a la aplicación a los datos privados de los usuarios. Más concretamente admite dos métodos para obtener el token de acceso a los datos: por código de autenticación o implícito.

Aunque para este tipo de aplicación ambos métodos serían válidos, se ha elegido usar la versión del protocolo que concede los tokens de acceso usando códigos de autenticación, puesto que éste requiere de dos interacciones para obtener los tokens, lo que consiguientemente le proporciona una capa más de seguridad.

Así pues, se ha implementado un mecanismo que permita al bot realizar el proceso del Oauth por código de autenticación. Este proceso empieza cuando se le proporciona al estudiante una dirección URL dónde puede autenticarse con su cuenta del Racó, para así dar acceso a sus datos a la aplicación. Esta parte del proceso se puede ver en la figura 6.

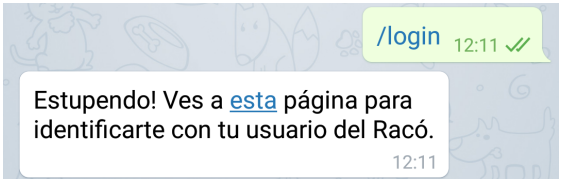

FIGURA 6: Resultado del comando /login.

Una vez el estudiante en cuestión se ha autenticado, la API del Racó redireccionará al estudiante a la dirección escogida por la aplicación, pasando como parámetro HTTP el código de autenticación.

Para hacer posible que el bot reciba el código, la redirección se hará a localhost (página inexistente), con el parámetro code insertado. Para que el bot pueda recuperar el código de autenticación, pregunta al usuario para que éste le proporcione la URL a la que se redirigió, y así poder extraer de ahí el código y terminar el proceso, como se puede ver en la figura 7.

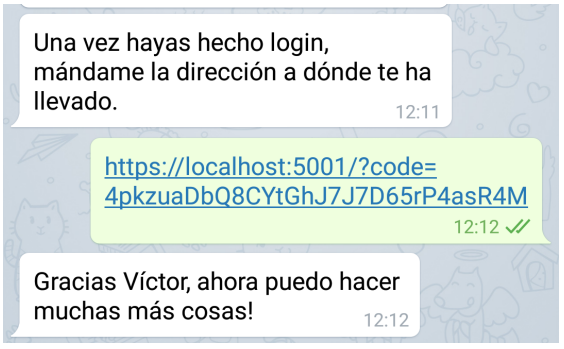

FIGURA 7: Segunda parte del proceso Oauth.

Es importante destacar que los códigos de autenticación tienen una duración de 10 minutos antes de caducar, por lo que si el usuario los no lo devuelve al bot pasado ese tiempo, se deberá volver a empezar el proceso.

El resultado final de todo este proceso es que el bot es capaz de obtener el token de acceso a los datos del usuario, siguiendo el diagrama presentado en la figura 8.

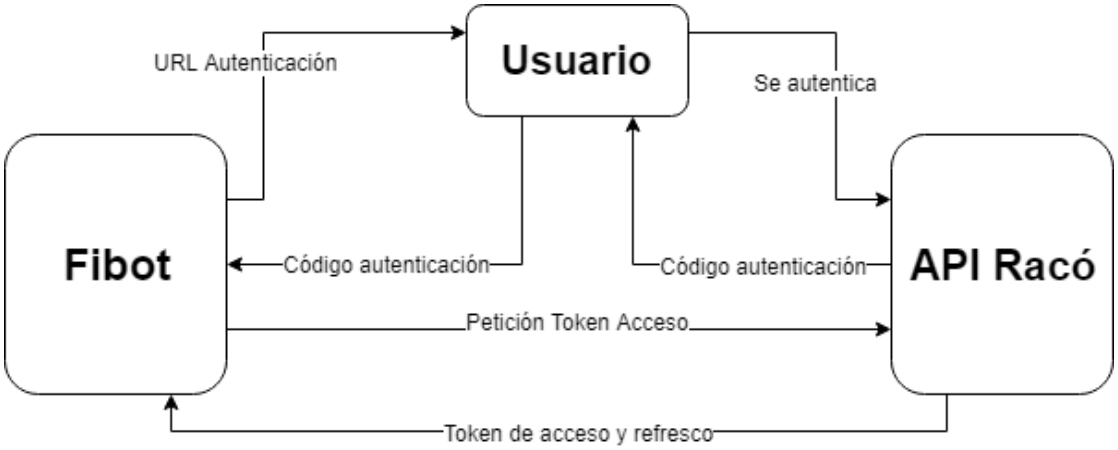

FIGURA 8: Diagrama del flujo del protocolo Oauth por código de autenticación.

Una vez se ha terminado el proceso y se ha obtenido el token de acceso y refresco para ese usuario, es necesario saber que éstos duran 10 horas antes de caducar, y pasado ese tiempo, para seguir teniendo acceso a los datos hace falta refrescar el token de acceso mediante el de refresco.

Por ello, el mecanismo encargado de obtener los tokens también implementa el método para refrescarlos usando los tokens de refresco.

7.2.2. Obtención de datos

Puesto que el objetivo principal del bot es proporcionar respuestas a consultas o preguntas, es muy importante que éste pueda obtener de forma rápida y eficaz la información por la que se le ha preguntado.

Aunque se podría haber diseñado un método genérico que dada una consulta estructurada fuera capaz de navegar por los datos de los que dispone la API del Racó para proporcionar la respuesta, se descartó implementar esta opción puesto que sería más costosa y provocaría un aumento en el tiempo de respuesta del bot, dado que su limitación principal es la latencia inherente que existe cuando se hacen peticiones HTTP.

De forma que para evitar tener ese problema, se han implementado métodos individuales para obtener la información concreta y necesaria para resolver las consultas sin la necesidad de navegar por la API hasta la información que se busca.

Estos métodos individuales permiten obtener:

- Horario de un estudiante.
- Asignaturas que un estudiante esté cursando.
- Exámenes de cualquier asignatura.
- Prácticas activas de un estudiante.
- Avisos sin caducar de un estudiante.
- Profesores (con sus correos electrónicos) de cada asignatura.
- Plazas libres y ocupadas para matricularse a una asignatura (para todos los grupos).

7.3. Multithreading

El bot hace uso de tres threads independientes para llevar a cabo distintas funcionalidades en paralelo. Éstos se corresponden a un thread principal, uno dedicado al refresco de tokens, y otro encargado de escanear notificaciones de los estudiantes.

7.3.1. Thread principal

El thread principal es el encargado de recibir, procesar y responder a los mensajes que recibe por parte de los usuarios. Esta parte se lleva a cabo siguiendo el proceso que se explica en el capítulo 8.

Para revisar si se han recibido mensajes nuevos se hace polling de forma constante, haciendo peticiones a la API de Telegram Messenger usando la función getUpdates descrita anteriormente.

Además, el thread principal es el thread encargado de recibir y procesar los comandos que el bot reciba. A continuación se lista el conjunto de comandos del bot.

- **/start**: Comando que inicia la conversación con el bot.
- **/login**: Comando que inicia el proceso de autenticación de mediante el protocolo Oauth tal como se ha descrito en la sección 7.2.1.
- **/logout**: Comando que elimina los tokens de acceso y refresco del usuario de la base de datos de usuarios, haciendo imposible el acceso a sus datos en la API del Racó.
- **/set_lang**: Comando que seguido del idioma escogido cambia el idioma en el que la conversación se llevará a cabo. Por ejemplo: set_lang ca, para cambiar el idioma de la conversación al catalán. Los lenguajes disponibles son 'es' para el castellano, 'en' para el inglés y 'ca' para el catalán.

- **/updates_on**: Comando que activa el envío de avisos de forma automática a los usuarios que se hayan identificado con su cuenta del Racó.
- **/updates_off**: Comando que desactiva el envío de avisos automáticos a los usuarios que los tuvieran activos.

La mayoría de los comandos están orientados a modificar los datos que se guardan de los usuarios, por lo que el thread principal también es el que gestiona la mayor parte de las interacciones con la base de datos de usuarios (excepto el refresco de tokens, tal como se explica en la sección 7.3.2).

Además, todos los cambios que efectúe el thread en la representación en memoria de la base de datos, se realizará también en los ficheros que conforman la base de datos para mantener siempre la coherencia entre la memoria y la base de datos.

Por último, el thread principal está pensado para seguir el comportamiento de una máquina de estados, aunque en la versión final ésta solo tiene dos estados. Los dos estados se corresponden a los almacenados en la base de datos de los usuarios, que son el de esperando pregunta/consulta por parte del usuario, y el de esperar al código de autenticación para terminar de hacer el *login*.

7.3.2. Thread de refresco de tokens

El thread de refresco de tokens es el que se encarga de actualizar los tokens de acceso de todos los usuarios antes de que caduquen y sean inservibles.

Para hacerlo, hace uso del polling, y en cada iteración revisa para todos los usuarios de la base de datos que se hayan identificado con la cuenta del Racó, si su token va a caducar. En caso positivo se usa el mecanismo descrito anteriormente para refrescar los tokens de acceso.

La frecuencia escogida para llevar a cabo el polling es cada minuto. Además para evitar que en el periodo entre dos iteraciones del polling pueda haber tokens caducados, en lugar de mirar si el token ha caducado para ser refrescado, se revisa si el token caducará en los próximos 5 minutos. De este modo en ningún momento puede haber tokens caducados.

7.3.3. Thread de escaneo de notificaciones

El thread de escaneo de notificaciones tiene como objetivo analizar todos los usuarios que se hayan identificado con su cuenta del Racó, y hayan activado las notificaciones. Su funcionamiento se limita a enviar mensajes de forma automática cuando hayan nuevos avisos, y además proporcionar información acerca del aviso.

El thread tiene en cuenta cuándo se recibió el último aviso, y así durante el proceso del polling se recogen todos los avisos de de cada uno de los estudiantes que tengan el servicio de notificaciones activado.

Estos avisos se filtran en función de si son posteriores al último recibido, y en caso de que lo sean se envía al estudiante uno o varios mensajes con información como qué asignatura tiene el aviso nuevo, qué titulo tiene, y si tiene archivos adjuntos.

Como en el caso anterior, el polling se hace cada minuto.

7.4. Gestor de mensajes

También se ha implementado un gestor de mensajes, que es capaz de mandar mensajes a cualquier usuario registrado en la base de datos de usuarios usando la API HTTP de Telegram, tal como se ha explicado en el capítulo 6.1.

El gestor de mensajes implementa las llamadas a la API con la posibilidad de mandar acciones, responder a mensajes anteriores, e interpretar los mensajes en formato Markdown para el envío de hiperenlaces, como se puede ver en la figura 6.

7.5. Modelo de comprensión del lenguaje (NLU)

Todo bot conversador debe ser capaz de comprender el lenguaje. En este caso, el bot que se desarrolla es un bot con propósito específico, y eso implica que su uso queda limitado a un dominio específico, y no para un dominio genérico. En este sentido, los mensajes tendrán una intención única, y consiguientemente, una respuesta correcta, cosa que no necesariamente ocurre en una conversación genérica.

Para que el bot pueda atender a los mensajes de un usuario, es necesario descubrir qué intención tiene cuando manda un mensaje, y si hay entidades en el mensaje que sean importantes para dar una respuesta correcta. Es decir, se debe hacer la siguiente transformación:

"Cuál es el correo de Javier Béjar?"

- **Intención**: Preguntar correo profesor
- **Entidades**: "Javier Béjar": nombre de profesor

7.5.1. El conjunto de datos

Primero se debe de generar un conjunto de datos que permita entrenar los modelos, que en este caso seguirán un aprendizaje supervisado (*Supervised learning*).

Este conjunto de datos debe contener preguntas y consultas como las que se quiere resolver, y debe indicar qué intención tiene cada pregunta o consulta, y qué entidades contiene.

Este conjunto de datos es generado de forma automática y balanceada. Eso significa que por cada posible intención que el bot contempla, habrá la misma cantidad de ejemplos. La generación automática se hace escogiendo aleatoriamente patrones para preguntas y consultas que se han preparado de forma manual, añadiendo, en caso de que sea necesario, entidades del tipo adecuado de forma aleatoria también. La información acerca de la generación de datos se localiza en la sección 7.9.2.

Además, en los datos se han definido 10 expresiones regulares que permitirán generar más características y mejoraran el rendimiento de los mecanismos encargados de la comprensión del lenguaje natural. Estas características están definidas para ayudar a descartar intenciones incorrectas.

Ejemplos de estas 10 expresiones regulares definidas son: "([0-9]{2} |grup. |group)" para identificar a grupos, y otras que buscan palabras clave como: "(mail |corre. |email)".

El dataset resultante de la generación aleatoria es un fichero json con los ejemplos siguiendo el formato mostrado en la figura 9.

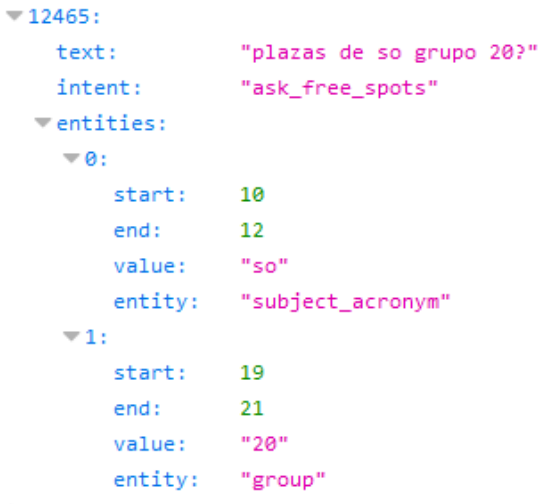

FIGURA 9: Un ejemplo de un dataset generado aleatoriamente.

El dominio considerado para el proyecto se presenta en la sección 7.6.1.

7.5.2. Tratamiento previo

Antes de generar cualquier modelo, es necesario procesar el conjunto de datos y obtener características de ellos para poder entrenar los modelos.

El primer paso de el tratamiento que reciben es la tokenización. Durante este proceso se segmentan los mensajes, y se obtienen tokens, usando el tokenizador de la librería spaCy [25].

El siguiente paso es obtener características del conjunto de tokens generados. Para el primero de los dos modelos que se van a crear se usan dos procesos. En primer lugar se usan las expresiones regulares definidas en los datos (ver sección 7.5.1). En esta parte del proceso se añadirán como características binarias si cada expresión se cumple o no en los datos.

A continuación se computará el context-sensitive tensor de cada token por cada ejemplo de los datos (más información acerca de los context-sensitive tensors en la sección 6.3). Posteriormente se promediaran para obtener un context-sensitive tensor que represente la frase entera. Así pues, lo que se añadirá como características será el vector en 384 dimensiones resultante de promediar estos tensores para cada frase de los datos. Estas 384 características serán numéricas.

Así pues, el primer modelo modelo tiene 394 características. 10 de las cuales son binarias e informan sobre qué expresiones regulares cumplen los datos, y las 394 restantes son numéricas y se corresponden al promedio de los context-sensitive tensors para cada ejemplo de los datos.

El segundo modelo requiere de calcular características individuales de los tokens, tales como si están en mayúsculas, minúsculas, si comienzan por mayúsculas, si contienen dígitos o de qué categoría gramatical son. El funcionamiento de éste se explica con más detalle en la sección 7.5.4.

7.5.3. *Intent Classification*

El proceso que consiste en predecir qué intención tiene un mensaje se denomina *Intent Classification*, y para llevarlo a cabo es necesario entrenar un clasificador que mediante las características definidas antes, decida qué intención es la correcta entre las 14 intenciones definidas en la sección 7.6.1.

En el proyecto, la técnica que se usa para el clasificador es una *Support Vector Machine* (SVM), que es una técnica que representa los datos como puntos en un espacio y separa las clases lo más ampliamente posible mediante uno o varios hiperplanos.

Para poder tener cierta flexibilidad, los SVM disponen de un hiperparámetro C que permite definir el margen para cometer errores durante el entrenamiento, creando así un margen blando (*soft margin*) que permita algunos errores en la clasificación a la vez que evite el sobre-ajustar o *overfitting*.

El SVM usa un kernel lineal y para decidir cuál es el mejor valor para el hiperparámetro C se usa *grid search Cross Validation* sobre un rango de posibles valores.

El resultado del modelo es un clasificador que dado un mensaje predice cuál es su intención proporcionando el grado de confianza de esa predicción. También proporciona el listado ordenado de clasificaciones alternativas por orden de confianza.

7.5.4. *Named Entity Recognition*

Las entidades nombradas (*named entities*) son secuencias que contienen nombres de personas, localizaciones, tiempo, cantidades, etc [26].

Named Entity Recognition (NER) es el proceso que consiste en reconocer o extraer éstas entidades nombradas.

Aunque librerías como spaCy ofrecen modelos entrenados sobre grandes cantidades de datos para reconocer entidades como lugares, nombres de persona, cantidades, tiempo, etc, éstas no necesariamente se corresponden con las entidades existentes en el ámbito académico, como siglas de asignaturas, o números de grupo, por lo que se ha descartado usarlas.

En su lugar, se usará una técnica de reconocimiento de entidades llamada NER-CRF (*Named Entity Recognition* usando *Conditional Random Field*). Un CRF es un modelo utilizado habitualmente para etiquetar y segmentar secuencias de datos o extraer información de documentos, que proporciona la flexibilidad de poder aprender entidades particulares de los datos.

Las características de los tokens (si están en mayúsculas, si son dígitos, etc), y el proceso de *Part-Of-Speech tagging* (POS tagging o etiquetado gramatical) dan probabilidades para ciertas clases de entidades, al igual que las transiciones entre etiquetas de clases vecinas. Finalmente se devuelve el conjunto más probable de clases.

El resultado final del modelo de comprensión puede verse en la figura 10, donde a la pregunta: *"Cuantas plazas hay de VC en el grupo 12?"*, el interprete reconoce con un 99.74 % de confianza la intención como una pregunta por plazas de matrícula, y también reconoce "VC" como unas siglas de una asignatura, y "12" como un grupo.

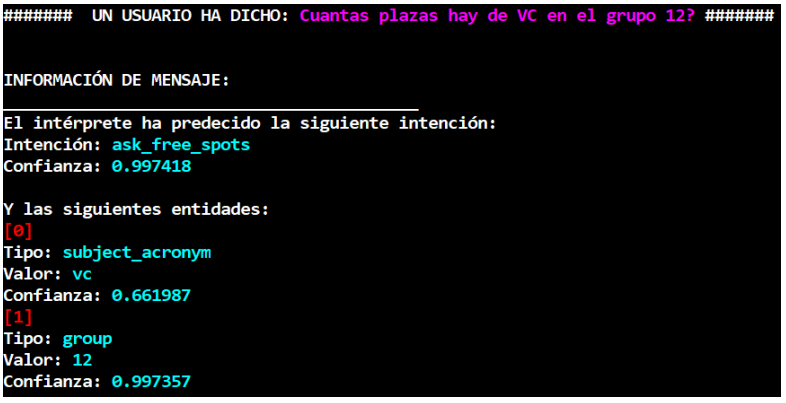

FIGURA 10: Resultado de aplicar el modelo de comprensión sobre una consulta.

7.6. Modelo de diálogo

La comprensión de un lenguaje es la primera parte de la capacidad de diálogo de un bot. También es necesario que aprenda, en este caso, qué acción debe llevar a cabo en cada contexto con la información que ha comprendido.

7.6.1. Dominio del diálogo

Puesto que el bot tiene un propósito específico, hace falta acotar cuál será el dominio en el cuál el bot puede comunicarse.

En este contexto, el dominio está formado por el conjunto de intenciones a las que debe ser capaz de responder, el conjunto de entidades que debe ser capaz de reconocer, qué elementos (*slots*) de la conversación debe recordar, y por último qué acciones tiene la posibilidad de tomar.

En primer lugar, el bot contempla 14 intenciones distintas:

- **greet**: Intención de saludar al bot.
- **thank**: Intención de agradecer al bot.
- **ask_teacher_mail**: Intención de conocer el correo electrónico de un profesor.
- **ask_teacher_office**: Intención de conocer la ubicación del despacho de un profesor.
- **ask_free_spots**: Intención de conocer la cantidad de plazas libres de una asignatura.
- **ask_subject_classroom**: Intención de conocer en qué aula se hacen clases de alguna asignatura.
- **ask_subject_schedule**: Intención de conocer el horario de alguna asignatura.
- **ask_subject_teacher_mail**: Intención de conocer el correo del profesor de alguna asignatura.
- **ask_subject_teacher_office**: Intención de conocer la ubicación del despacho del profesor de alguna asignatura.
- **ask_subject_teacher_name**: Intención de conocer la el nombre de los profesores de alguna asignatura.
- **ask_next_class**: Intención de conocer detalles acerca de su próxima hora de clase.
- **ask_exams**: Intención de conocer acerca de sus próximos exámenes.
- **ask_pracs**: Intención de conocer acerca de sus próximas prácticas abiertas.
- **inform**: Intención de aportar información para terminar una consulta, aunque también se usa para ser informado acerca de un profesor o asignatura.

En lo referente a las entidades que debe poder reconocer existen 3:

- **teacher_name**: Nombre (completo o incompleto) de una persona.
- **subject_acronym**: Siglas de una asignatura.
- **group**: Indicador numérico de un grupo de una asignatura.

Para el proyecto se han definido 7 *slots*, 3 de los cuales coinciden con las entidades puesto que deben ser recordadas a lo largo de la conversación. Los siguientes 4 son:

- **matches**: *Slot* booleano que indica si se requiere de más información para responder a una consulta.
- **subject_existance**: *Slot* booleano que indica si las siglas de asignaturas almacenadas en su *slot* se corresponden a una asignatura existente en la API del Racó.
- **subject_enrollment**: *Slot* booleano que indica si el usuario está matriculado a la asignatura cuyas siglas están en el *slot* correspondiente.
- **user_logged**: *Slot* booleano que indica si el usuario se ha identificado con su usuario del Racó.

Finalmente, se han definido 18 acciones, que se explican en detalle en la sección 7.7.

7.6.2. Historias

El siguiente paso para obtener un modelo de diálogo es generar el conjunto de datos. En este caso, el conjunto de casos viene representado por lo que se llaman historias. Éstas definen vías en las que puede suceder la conversación, y representan los posibles estados de la conversación que pueden ocurrir. Las historias servirán para entrenar el modelo que permitirá al bot decidir qué acciones tomar al interpretar los mensajes.

Como se puede ver en el ejemplo de la figura 11, la historia define cuál sería un posible comportamiento del bot cuando se le pregunta por el correo del profesor de una asignatura que tiene múltiples profesores.

```
## Generated Story subject teacher mail 1
* ask_subject_teacher_mail{"subject_acronym": "prop"}
    - Action_check_subject_existance
    - slot{"subject_existance": true}
    - Action_show_subject_teachers_mails
    - slot{"matches": true}
* inform{"teacher_name": "jordi turmo"}
    - Action_show_teacher_mail
    - Action_slot_reset
```

FIGURA 11: Ejemplo de historia.

En este caso, las historias se han generado de forma totalmente manual considerando el dominio del diálogo al completo.

7.6.3. El modelo

Para generar el modelo de dialogo que sea capaz de predecir qué acción debe realizarse teniendo en cuenta el estado actual de la conversación se parte de las historias como datos.

Las historias definidas anteriormente permiten recrear diferentes estados de la conversación. Un estado viene representado por:

- La última acción realizada
- Qué intención y qué tipo de entidad se han identificado en el mensaje
- Qué slots están definidos

Toda la información que representa un estado se codifica teniendo en cuenta la presencia o ausencia de información en un punto de la conversación. Por lo que se representa como vectores binarios codificando las distintas categorías, es decir, las distintas acciones, intenciones, entidades y slots usando *one hot encoding*.

Con esta información vectorizada, se debe predecir la siguiente acción a tomar. Para ello, se usa una red neuronal recurrente (RNN) como clasificador, disponiendo de células *Long Short-Term Memory* (LSTM).

Una RNN es en cierta forma parecida a una red neuronal clásica, pero con la posibilidad de recibir como entrada, elementos que se han percibido en momentos anteriores, lo que las hace una muy buena opción para manejar situaciones secuenciales como los diálogos.

Se usa este tipo de red porque las LSTM dan buenos resultados en modelos donde hay información que debe ser "recordada" por el modelo. En el caso concreto de la aplicación, otras arquitecturas proporcionarían buenos resultados dado que las posibles interacciones con el bot son cortas y no hay un beneficio claro al usar LSTM, pero una arquitectura como esta facilitaría posibles extensiones futuras.

Finalmente, la arquitectura de la red consiste en:

- Una capa de entrada con 42 neuronas, correspondientes a la representación del estado comentada anteriormente.
- Una capa oculta con 32 células LSTM.
- Una capa de salida con 18 neuronas, correpondientes a cada una de las posibles acciones.

7.7. Acciones

Las acciones son los elementos que hacen que el bot sea capaz de proporcionar respuestas al usuario. En el caso del bot propuesto para el proyecto, las acciones permitirán obtener los correos de profesores, averiguar horarios de asignaturas, etc. Toda la información resultante de las acciones se devuelve ya en forma de mensaje en lenguaje natural tal como se explica en la sección 7.8.

Tal como se ha explicado en la sección 7.6.1, existen 18 acciones definidas en el dominio, que son las siguientes:

- **Action_greet**: Acción que saluda al usuario.
- **Action_no_problem**: Acción que atiende a los agradecimientos de los usuarios.
- **Action_show_teacher_mail**: Esta acción pretende devolver un mensaje con la dirección de email del profesor con nombre más similar al que el bot que esté en el estado.
- **Action_show_teacher_office**: Esta acción tiene la misma finalidad que la anterior, pero haciendo referencia al despacho del profesor en lugar del correo electrónico.
- **Action_show_subject_free_spots**: Esta acción devuelve la cantidad de plazas libres para matricularse en la asignatura cuyas siglas estén en el estado de la conversación y filtra por grupo en caso de que haya un grupo presente en el estado.
- **Action_show_subject_classroom**: Esta acción pretende informar acerca de en qué aulas se llevan a cabo las clases de la asignatura cuyas siglas están almacenadas en el estado.
- **Action_show_subject_schedule**: Acción similar a la anterior, pero proporciona información de cuándo se tiene clase de la asignatura registrada en el estado.
- **Action_show_subject_teachers_mails**: Esta acción espera disponer de siglas de alguna asignatura en el estado y proporciona los correos de los profesores de la asignatura. En el caso en que la asignatura tenga más de 4 profesores proporcionará el listado de profesores para poder concretar en interacciones posteriores, como se puede ver en la figura 12.
- **Action_show_subject_teachers_offices**: Esta acción tiene el mismo comportamiento que la anterior, pero en lugar de proporcionar los correos de los profesores proporciona los despachos.
- **Action_show_subject_teachers_names**: De forma muy similar a las dos acciones anteriores, ésta proporciona el listado de nombres de los profesores de la asignatura guardada en el estado de la conversación.

- **Action_show_next_class**: Esta acción devuelve un mensaje con información de qué día es la siguiente clase del alumno, es decir de qué asignatura, a qué hora y en qué aula.
- **Action_show_next_exams**: Acción que proporciona información acerca de los exámenes de los siguientes 2 meses. Si en el estado de la conversación figuran las siglas de alguna asignatura, se filtrarán los resultados.
- **Action_show_next_pracs**: Esta acción proporciona información acerca de las prácticas abiertas de los siguientes 2 meses. Si en el estado de la conversación figuran las siglas de alguna asignatura, se filtrarán los resultados.
- **Action_show_teacher_info**: Acción que proporciona información acerca del profesor cuyo nombre está almacenado en el estado de la conversación.
- **Action_slot_reset**: Esta acción provoca que el agente que gestiona la conversación olvide la información que ha registrado en los *slots*.
- **Action_check_subject_existance**: Esta acción comprueba que haya una asignatura en el estado, y si ésta existe en la API.
- **Action_check_subject_enrollment**: Acción que comprueba si el usuario está matriculado a la asignatura del estado.
- **Action_check_user_logged**: Acción que comprueba si un usuario se ha identificado con su cuenta del Racó.

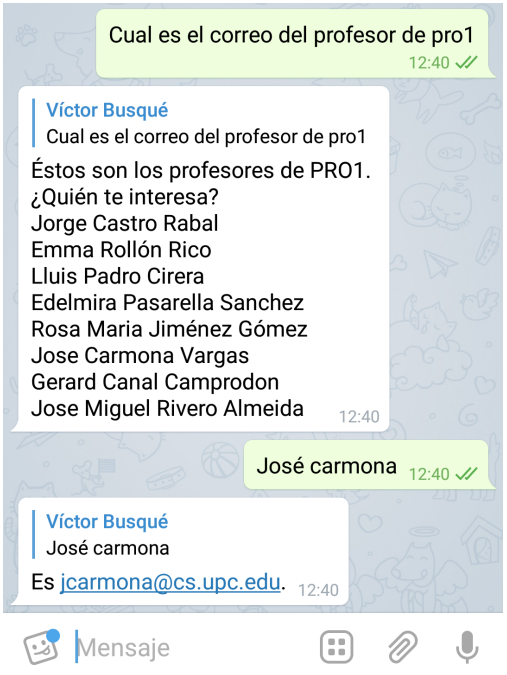

FIGURA 12: Diálogo con dos interacciones.

El conjunto de acciones implementadas permite de forma fácil la gestión de errores o información incorrecta, como se puede ver en la figura 13.

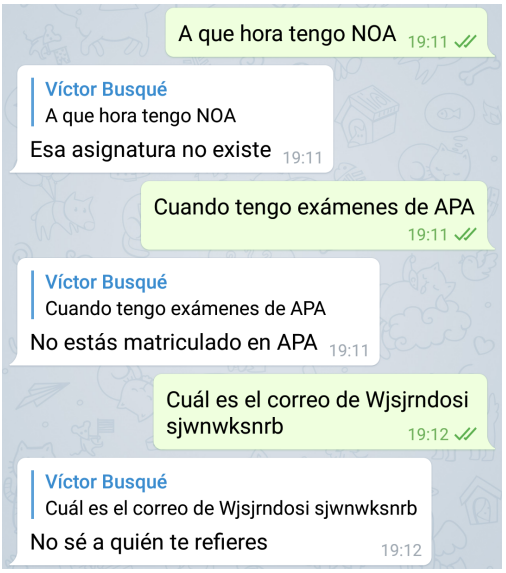

FIGURA 13: Respuestas a casos excepcionales.

Finalmente, como se ha mencionado, las acciones dan como respuesta un mensaje escrito en lenguaje natural. Para generar este mensaje se parte de plantillas generadas de forma manual, que permiten ser completadas con la información necesaria.

Para hacerlo, se han almacenado en un fichero json múltiples formas de contestar a una pregunta, como se puede ver en la figura 14.

```
"ask_next_pracs": {
  "ca": [
  "es": [
    "Tienes la práctica '{0}' de {1} para el día {2} de {3} a las {4}h.",
    "Debes entregar la práctica {0} de {1} el {2} de {3} antes de las {4}h."
  ],
```

FIGURA 14: Ejemplo de plantilla para respuestas.

7.8. Multi-idioma

Para poder ser útil a una mayoría de estudiantes de la facultad, el bot debe ser capaz de comprender y responder en catalán, castellano e inglés.

En primer lugar, para que el bot sea capaz de comprender los tres idiomas se deben generar tres modelos de comprensión del lenguaje natural. Es por ello que se crean tres *datasets*, cada uno conteniendo ejemplos de preguntas en uno de los idiomas.

Los tres datasets contemplan la misma cantidad de intenciones y entidades, por lo que los tres idiomas podrán obtener la misma información. El proceso de creación de los datasets está descrito en la sección 7.9.2.

Con cada dataset se entrena un modelo distinto, y por eso, independientemente del idioma en el cual se pregunte, se terminará prediciendo una intención del mensaje, y si hay entidades.

En cuanto al modelo de diálogo, no es necesario llevar a cabo ningún cambio, porque la información que recibe, es decir, la intención y las entidades del mensaje, y el estado de la conversación, no depende del idioma en el que ha sido interpretado el mensaje.

Finalmente, para poder proporcionar una respuesta en el idioma correcto al usuario, se prepararan múltiples plantillas para mensajes de respuesta en cada idioma con el objetivo de poder responder a las preguntas, tal como se ha mostrado en la figura 14.

7.9. Elementos auxiliares

Además de todos los elementos que conforman el bot, también se han desarrollado herramientas externas que hacen posible el funcionamiento del bot.

7.9.1. Scraper del Directori UPC

El Scraper del Directori UPC es la herramienta encargada de recolectar la información acerca de los profesores que se ha comentado en la sección 7.1.2.

La información del Directori UPC está organizada por unidades, es decir, por departamentos identificados por un código. Por ejemplo, el departamento de Ciencias de la Computación tiene el código 723, y accediendo a la dirección `http://directori.upc.edu/directori/dadesUE.jsp?id=723` se puede obtener su información, como muestra la figura 15.

FIGURA 15: Departamento de CS en el Directori UPC.

Cada departamento dispone del listado de personal asociado, que da acceso a la información de cada docente, como se puede ver en la figura 16

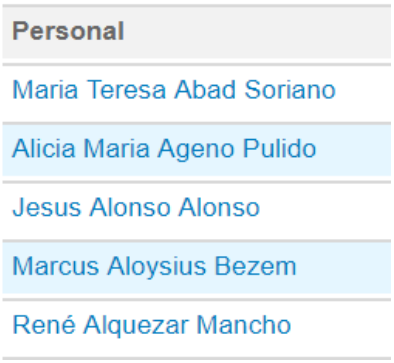

FIGURA 16: Fragmento del listado de personal del departamento de CS.

De la misma forma que las unidades o departamentos, cada persona del listado de personal dispone de su código identificativo a través del cual se puede acceder a su información. Por ejemplo, en el caso concreto del director del proyecto, Javier Béjar, el identificador como personal docente es 1001970, y accediendo a la dirección: `http://directori.upc.edu/directori/dadesPersona.jsp?id=1001970`, se puede obtener la información que se muestra en la figura 17.

FIGURA 17: Información de un docente.

Visto el funcionamiento del Directori UPC, a continuación se explicará el funcionamiento del scraper implementado.

El scraper recibe como entrada los códigos identificativos de los departamentos que se desean analizar. Con esa información, se recorren los departamentos accediendo al listado de personal asociado e indexando los códigos identificativos del personal.

Posteriormente, por cada código identificativo indexado, se accederá a la página que contiene la información del personal docente, lo que proporcionará como resultado el código de la página en HTML. Con éste, se llevará a cabo un análisis para extraer la información necesaria que se pretende almacenar, es decir, el correo electrónico y el despacho. Éste análisis se hace usando expresiones regulares presentes en el código HTML para poder identificar y extraer la información mencionada.

La información que se ha registrado al final del proceso, se organiza por departamentos y se almacena en formato json.

7.9.2. Generador de datasets para el NLU

Se ha implementado un generador de datasets para poder entrenar el NLU en los tres idiomas que contempla el proyecto. Un dataset consiste en un fichero json por idioma que contiene ejemplos de preguntas etiquetadas con la intención correcta y las entidades presentes.

Para generar estos ficheros, se parte de un conjunto de formas de preguntar correspondiente a cada intención, introducidas manualmente. Estas formas o plantillas se han introducido en un fichero json, como se puede ver en la figura 18.

```
"es": {
  "intros_teacher_mail": [
  "intros_teacher_desk": [
    "cual es el despacho de {}",
    "despacho de {}",
    "donde esta el despacho de {}",
    "dime el despacho de {}",
    "dime donde esta el despacho de {}",
    "me puedes decir donde esta el despacho de {}",
    "dime cual es el despacho de {}",
```

FIGURA 18: Fragmento del listado de plantillas.

Además de el conjunto de plantillas manualmente introducidas, también se dispone de un listado de nombres de profesores, y de un listado de siglas de asignaturas, para poder poblar los huecos de los que disponen las plantillas con entidades del tipo correspondiente. En el caso de entidades correspondientes a nombres de profesores, se han considerado nombres completos, o parciales (nombre y apellido, sólo nombre).

El generador de datasets recibe éstos elementos, y también la cantidad de ejemplos a generar por cada tipo de pregunta, es decir, intención. Con esta información, generará ejemplos escogiendo de forma aleatoria plantillas para cada intención, y poblando los huecos de los que dispongan con ejemplos aleatorios de las entidades necesarias.

Dos ejemplos de un dataset generado aleatoriamente se pueden ver en la figura 19.

```
{
  "text": "dime donde esta el despacho de javier bejar alonso",
  "intent": "ask_teacher_office",
  "entities": [
    {
      "start": 31,
      "end": 50,
      "value": "javier bejar alonso",
      "entity": "teacher_name"
    }
  ]
},
{
  "text": "despacho de ilario bonacina",
  "intent": "ask_teacher_office",
  "entities": [
    {
      "start": 12,
      "end": 27,
      "value": "ilario bonacina",
      "entity": "teacher_name"
    }
  ]
},
```


FIGURA 19: Ejemplos de un dataset generado.

7.9.3. Entrenador de los modelos

El entrenador de modelos es una pequeña herramienta implementada para poder entrenar los tres modelos de comprensión y el modelo de gestión de diálogo.

7.9.4. Herramienta de test del NLU

Durante el desarrollo del proyecto, se ha considerado útil implementar una pequeña herramienta capaz de probar el comportamiento del NLU. Además, esta herramienta será de gran utilidad para la experimentación y resultados acerca del NLU.

La herramienta permite testear de forma manual introduciendo preguntas y obteniendo el resultado proporcionado por el NLU, y también permite usar como entrada ejemplos desde un fichero.

Cuando no hay más ejemplos para interpretar, la propia herramienta proporciona la matriz de confusión de los resultados interpretados, así como la precisión y la exhaustividad de cada intención.

Además, calcula el promedio de la confianza cuando acierta y cuando falla en clasificar las intenciones, y permite la observación de los errores de clasificación.

Capítulo 8

Pregunta-Respuesta

Cuando el bot recibe un mensaje por parte de un usuario se llevan a cabo múltiples procesos hasta proporcionar la respuesta. En este capítulo se detallará el protocolo a seguir para procesar un mensaje. Para que sea más intuitivo se usará como ejemplo el mensaje:

"Cuál es el correo de Javier Béjar?"

8.1. Procesado del mensaje

El primer paso antes de que el mensaje sea procesado por cualquier modelo es obtener características de él. Por este motivo, el mensaje sufrirá el mismo tratamiento que los datos de entrenamiento del NLU, es decir, se tokenizará, y posteriormente se recopilarán características de él comprobando cuáles de las expresiones regulares definidas en los datos cumple y el promedio de los vectores (*averaged words*), tal como se ha explicado en la sección 7.5.2. Esta parte del proceso se resume en el diagrama disponible en la figura 20.

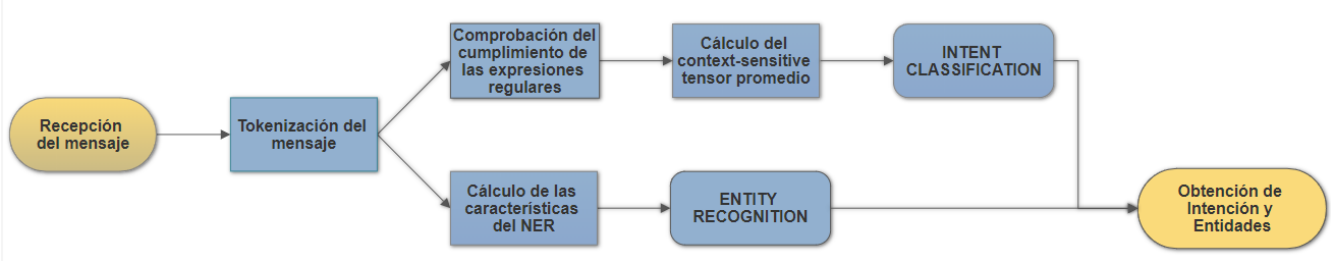

FIGURA 20: Diagrama del procesado del NLU al mensaje.

Estas características serán usadas en el siguiente proceso, que consiste en llevar a cabo el *Intent Classification* y la *Named Entity Recognition*. El resultado de ésto se puede ver en la figura 21.

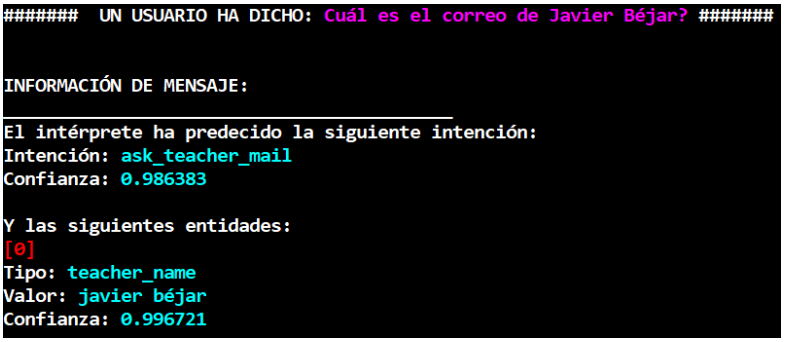

FIGURA 21: Ejemplo del procesado del NLU al mensaje.

8.2. Filtrado intermedio

Dado que el bot está expuesto a recibir cualquier mensaje, necesita disponer de algún filtro que le permita descartar mensajes que no encajen en el dominio del diálogo, es decir, que no tengan alguno de los propósitos definidos.

El mecanismo que se ha escogido para decidir cuándo descartar un mensaje y cuando no se basa en aprovechar el grado de confianza que proporciona el clasificador de intenciones. Se ha establecido que si el grado de confianza en la predicción de la intención es inferior al 50 % el mensaje no se ajusta a ninguno de los propósitos contemplados, por lo que no es necesario procesarlo y proporcionar una respuesta, puesto que la intención con más confianza probablemente es aleatoria.

En tal caso se devolverá un mensaje indicativo de que no se ha comprendido la pregunta. Como se puede ver en la figura 22, se responde con un mensaje indicando que no se ha comprendido puesto que el mensaje enviado se clasifica como "inform" con una confianza de 41.22 %.

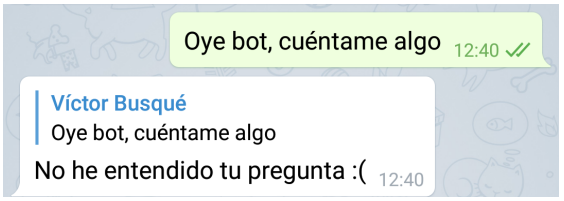

FIGURA 22: Filtrado de un mensaje por confianza.

8.3. Procesado del agente

En caso de que el mensaje no sea descartado en el filtraje, el siguiente paso es predecir cuál es la siguiente acción a tomar.

En este punto se recoge la intención y las entidades interpretadas en el primer paso, y juntamente con el estado actual de la conversación (*slots* recordados y acciones tomadas) el modelo de gestión de dialogo predirá la siguiente acción a tomar.

En caso de que la siguiente acción a tomar deba de retornar un mensaje al usuario se llevará a cabo el envío del mensaje, y posteriormente se procederá a actualizar el estado de la conversación con la acción tomada.

Un diagrama que resume el funcionamiento del bot en esta fase se puede ver en la figura 23.

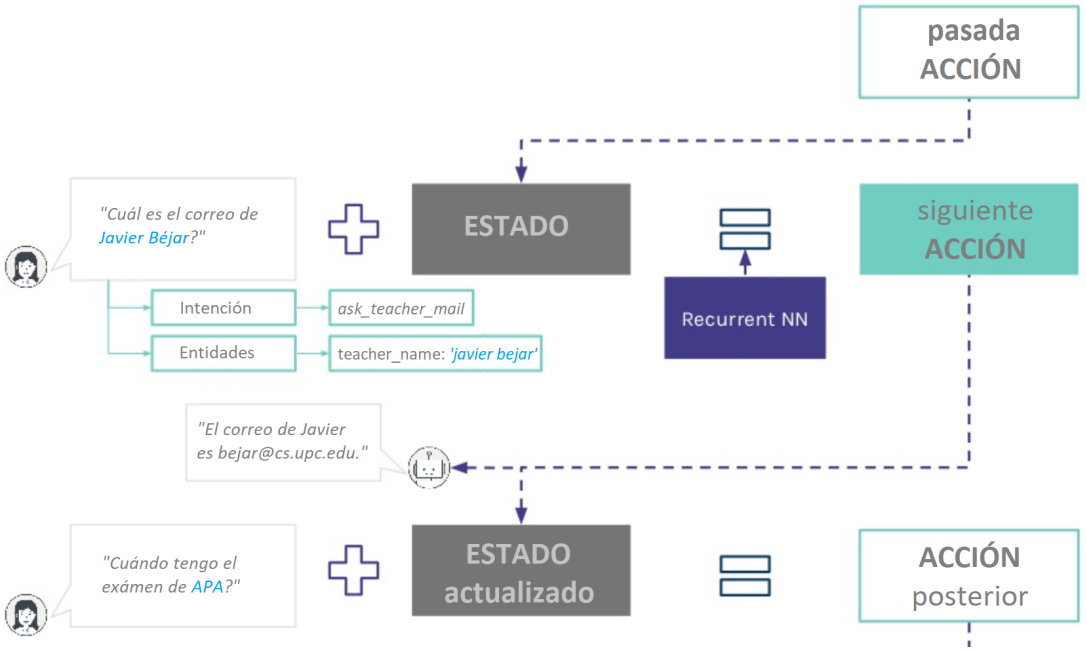

FIGURA 23: Diagrama del funcionamiento del gestor de diálogos.

En el caso concreto de la pregunta que se ha realizado, el procesado que se hace puede verse en la figura 24.

```
Ejecutando acción:      Action_show_teacher_mail
Representación de los slots:
        group: None
        matches: None
        subject_acronym: None
        subject_enrollment: None
        subject_existance: None
        teacher_name:  javier béjar
        user_logged: None
Cargados en memoria los profesores de los departamentos
Distancia al profesor más cercano: 1
El profesor más parecido es: javier bejar alonso.

Ejecutando acción:      Action_slot_reset
Reseteando todos los slots...

Enviando mensaje... El correo de Javier es bejar@cs.upc.edu.
```

FIGURA 24: Toma de decisiones para el ejemplo.

Capítulo 9

Experimentación y resultados

9.1. Experimentación

El elemento más importante del proyecto es el modelo de comprensión del lenguaje natural (NLU). En esta sección se analizará cómo varios elementos afectan al rendimiento del modelo de NLU para obtener un buen modelo.

9.1.1. Experimentación del clasificador de intenciones

En primer lugar se comprobará como afecta el tamaño del conjunto de datos a los resultados que proporciona el modelo NLU para clasificar intenciones. Es importante tener en cuenta que toda la experimentación que se ha documentado es la experimentación con SVMs usando kernel lineal, pero los resultados usando otros kernels como rbf o polinomial no mejoran el rendimiento del lineal.

Como se ha comentado en la sección 7.5.1, el conjunto de datos se genera de forma aleatoria, pero con la misma cantidad de datos por cada intención. Para que la experimentación sea consistente, todos los datos sobre los que se van a entrenar los modelos serán generados por la misma semilla.

Durante toda la experimentación del modelo de NLU se usará un banco de pruebas que ha sido preparado de forma manual por 3 alumnos de la facultad y contiene 286 preguntas y consultas etiquetadas con las intenciones apropiadas.

Debido a que el conjunto de datos de entrada al modelo se genera de forma aleatoria, es posible que algunas preguntas del banco de pruebas coincidan con preguntas que existan en el conjunto de datos sobre el cual se ha entrenado el modelo. Además, aunque el banco de pruebas está en castellano, se ha comprobado que los resultados en catalán e inglés siguen un comportamiento muy parecido.

La tabla 10 muestra los resultados de la experimentación y proporciona la precisión, es decir, cuantas veces la intención correcta es la predicha con mayor confianza. También proporciona la confianza de la opción escogida cuando acierta y cuando falla.

datos por intención	**precisión**	**confianza aciertos**	**confianza fallos**
25	0.769	0.599	0.357
50	0.849	0.739	0.419
100	0.881	0.845	0.517
250	0.891	0.908	0.547
500	0.898	0.933	0.537
1000	0.905	0.934	0.545

Tabla 10: Resultados de experimentación del NLU en castellano.

El objetivo de esta parte de la experimentación es definir cómo debe ser el modelo para tener un funcionamiento óptimo. Más concretamente, lo que se espera del modelo es que tenga la mayor precisión posible, pero además se espera que haya la máxima diferenciación en confianza cuando acierta en la predicción respecto cuando falla. Esta última característica ayudará al bot a poder diferenciar cuando ha acertado y cuando no, y consiguientemente permitirá realizar mejor el filtrado intermedio del procesado del mensaje.

La métrica escogida para valorar qué kernel se usará y cuántos datos van a usarse para entrenar el modelo se decidirá en función de:

$$metrica = precision \cdot (\frac{confianza\ aciertos}{confianza\ fallos})$$

Como se puede ver en la tabla 11, el caso en el se obtiene la mejor puntuación de la métrica es cuando se usan 500 ejemplos de preguntas y consultas por cada intención.

datos por intención	**métrica**
25	1.290
50	1.497
100	1.440
250	1.479
500	1.560
1000	1.551

Tabla 11: Métrica en función de los datos.

9.1.2. Experimentación del extractor de entidades

En la experimentación del extractor de entidades se han analizado los resultados del reconocimiento de entidades definiendo diferentes características a tener en cuenta al entrenar, el límite de iteraciones que se usarán para los algoritmos de optimización de *Sklearn-CRFSuite* (que es la librería usada para implementar el NER-CRF) y la cantidad de datos sobre los que se entrena el modelo de NER.

Para explicar la experimentación se empezará analizando cómo se comporta el extractor de entidades en función de las características que calcula al entrenar. Estas características permiten definir qué elementos debe tener en cuenta el modelo acerca de la palabra que se está analizando, la palabra anterior y la siguiente para decidir qué clase de entidad la palabra en cuestión es. Éstas son:

- Si la palabra está escrita en mayúsculas
- Si la palabra está escrita en minúsculas
- Si la palabra empieza por mayúsculas
- Categoría gramatical de la palabra
- Si la palabra contiene dígitos
- Cantidad de palabras que forma la entidad

En esta parte de la experimentación se ha comprobado que decidir el tipo de entidad teniendo en cuenta si la palabra está escrita en mayúsculas o minúsculas no tiene un gran efecto en el rendimiento del extractor, pero contribuye a predecir mejor los nombres de profesores cuando éstos empiezan en mayúsculas, y a predecir correctamente siglas de asignaturas cuando están escritas completamente en mayúsculas.

Además, la categoría gramatical de las palabra en concreto y las de su entorno permiten identificar bastante bien qué tipo de entidad es.

El hecho de que la palabra contenga dígitos permite diferenciar muy bien cuando se trata de grupos o de otro tipo de entidades.

Por último, la cantidad de palabras que forma la entidad ayuda a definir bien cuando una entidad es el nombre de un profesor si éste tiene dos o más palabras, pero a su vez penaliza el caso en que se proporcione solo una palabra como nombre de el profesor.

Se ha valorado que es más beneficioso predecir bien el nombre de un profesor cuando tiene más de una palabra, y en su defecto equivocarse cuando sólo se proporciona una, debido a que usualmente con una sola palabra no siempre es posible identificar al profesor por el que se pregunta.

En la experimentación respecto al número máximo de iteraciones para los algoritmos de optimización se han probado distintos valores y se ha analizado el comportamiento del NER.

Para valores bajos (5 iteraciones), el NER tan sólo es capaz de reconocer algunas siglas de asignaturas, pero no en todos los contextos.

Para valores a partir de 20 el NER reconoce todos los grupos correctamente. Acerca de las siglas de asignaturas y los nombres de profesores detecta la inmensa mayoría correctamente, pero en ocasiones muy poco frecuentes hay ciertas oraciones que provocan que se detecten entidades que no debería.

Finalmente, se ha experimentado sobre la cantidad de datos y se ha visto como con pocos datos (entre 15 y 50 preguntas por intención), en general el modelo era capaz de identificar correctamente la mayoría de intenciones, pero en bastantes ocasiones no identificaba entidades en consultas simples como: *"Hay sitio en VJ subgrupo 22?"*, en la que es capaz de identificar "22" como grupo pero no "VJ" como siglas de una asignatura.

A partir de 100 preguntas por intención, el modelo detecta correctamente la inmensa mayoría de las entidades presentes en las preguntas. En algunas ocasiones muy concretas comete errores, por ejemplo:

"Cual es la oficina del profesor Javier Bejar"

En este caso se reconoce "profesor javier bejar" como el nombre de un profesor. Esto puede deberse a que "profesor" tiene la misma categoría gramatical que un nombre de profesor, y además el modelo se ha entrenado con frases similares a "*Cual es la oficina de Javier Bejar*", y con la información con la que cuenta el modelo, "profesor javier bejar" es un buen candidato a ser un nombre de profesor.

"Correo de Bejar"

Este ejemplo es un caso común de error del reconocedor de entidades, en el que reconoce "bejar" como las siglas de una asignatura. Esto ocurre cuando se refiere a un profesor con una sola palabra. Aunque el modelo ha sido entrenado con ejemplos de nombres de profesor formados por una palabra o múltiples, debido a que hay una presencia muy superior de ejemplos de siglas de asignaturas, el modelo tiene cierta preferencia en predecir ese tipo de entidad.

9.1.3. Experimentación del modelo de diálogo

En este caso no ha sido necesaria mucha experimentación sobre el modelo de diálogo, pues el modelo entrenado con los parámetros por defecto es capaz de seguir el procesado por las acciones de forma correcta en todos los casos. Eso significa que en los casos en los que el modelo de NLU detecta correctamente las intenciones y entidades, el modelo de diálogo siempre es capaz de gestionar qué acciones deben llevarse a cabo.

Esto se debe a que las normas y formas en las que puede suceder el diálogo son sencillas y no hay muchas interacciones que requieran de múltiples pasos.

9.2. Resultados

A nivel de resultados se comenzará comentando los resultados del modelo de comprensión del lenguaje. A continuación, en la tabla 12 se presenta la matriz de confusión del clasificador de intenciones usando el modelo entrenado según la experimentación sobre el banco de pruebas comentado.

Intenciones reales	**Predicciones**													
ask_teacher_mail	20	0	0	0	0	0	0	0	0	0	0	0	0	0
ask_teacher_office	0	20	0	0	0	0	0	0	0	0	0	0	0	0
ask_free_spots	0	0	20	0	0	0	0	0	0	0	0	0	0	0
ask_subject_classroom	0	0	0	15	2	0	1	0	0	1	1	0	0	0
ask_subject_schedule	0	0	0	1	22	0	0	0	0	0	1	0	0	0
ask_subject_teacher_mail	1	0	0	0	0	19	0	1	0	0	0	0	0	0
ask_subject_teacher_office	0	0	0	0	1	0	18	2	0	0	0	0	0	0
ask_subject_teacher_name	0	0	0	1	0	0	0	20	0	0	0	0	0	0
ask_next_class	0	0	0	4	1	0	0	0	18	0	0	0	0	0
ask_exams	0	1	0	0	1	0	1	0	0	17	0	0	0	0
ask_pracs	0	0	0	0	0	0	0	0	0	0	21	0	0	0
inform	0	1	0	0	0	0	0	0	0	0	0	25	0	0
greet	0	0	0	0	0	0	0	0	0	0	0	2	11	2
thank	0	0	0	0	0	0	0	0	0	0	0	4	0	10

Tabla 12: Matriz de confusión del modelo final en castellano

Como se puede deducir de la tabla, modelo rinde muy bien, especialmente cuando las preguntas son sencillas y directas. Para preguntas más complicadas donde hay menos información relevante (es decir, preguntas dónde un pequeño porcentaje de las palabras indica la intención) tiene a fallar más, y de ahí surgen algunas de las confusiones que existen en la matriz.

Además, es importante decir que un 60.7 % de las veces que el clasificador de intenciones falla, la segunda intención con más confianza es la correcta.

En general, el uso de expresiones regulares distintivas como características ayuda a mejorar la precisión en gran forma, y por eso intenciones como "ask_pracs" rinden tan bien aun con preguntas largas y rebuscadas, porque la expresión regular reduce el error en ese tipo de clasificación.

Un fenómeno interesante observado, es que en mensajes muy cortos (correspondientes a las intenciones de "greet" o "thank") suele haber confusiones junto con "inform". Esto se debe a que el modelo se ha entrenado con saludos y agradecimientos simples y directos como: "gracias", "muchas gracias", "hola" o "buenos días", pero en el banco de pruebas se ha encontrado con frases como "te lo agradezco de verdad", lo cual difiere mucho de la representación de agradecimiento entrenada.

Siguiendo con los resultados del modelo de comprensión del lenguaje natural, se analizarán los resultados del extractor de entidades. En general, el modelo detecta muy bien las entidades para preguntas o consultas simples e incluso más elaboradas, dónde normalmente existe un cierto patrón sobre el cual se construye la frase, por ejemplo: *"cual es el mail de [nombre profesor]"* o *"Podrías decirme dónde puedo encontrar el despacho de [nombre profesor]"*. El modelo detecta fácilmente que lo que hay después de la palabra "de" es una entidad, y en función de sus características decide qué tipo de entidad es.

Usualmente eso es beneficioso, pero en otros casos, por ejemplo, *"Me puedes decir donde se encuentra ubicado el despacho de mi profesor de IA"*, el extractor de entidades detecta correctamente "IA" como asignatura, pero también la palabra "mi" como asignatura.

A nivel de gestión de diálogo, el modelo ha sido capaz de asimilar qué comprobaciones debe hacer en cada caso para asegurarse de dar la respuesta adecuada. Por ejemplo, ha aprendido que cuando se le pregunta por las prácticas de alguna asignatura en concreto, debe revisar que el usuario se haya identificado con su cuenta del Racó, que la asignatura exista y que el usuario esté matriculado en la asignatura antes de proporcionar alguna respuesta, como se puede ver en la figura 25.

Además, en los casos en los que se requiere de una segunda interacción por parte del usuario, el modelo es capaz de seguir con la conversación en caso de que la información requerida en la segunda interacción sea proporcionada.

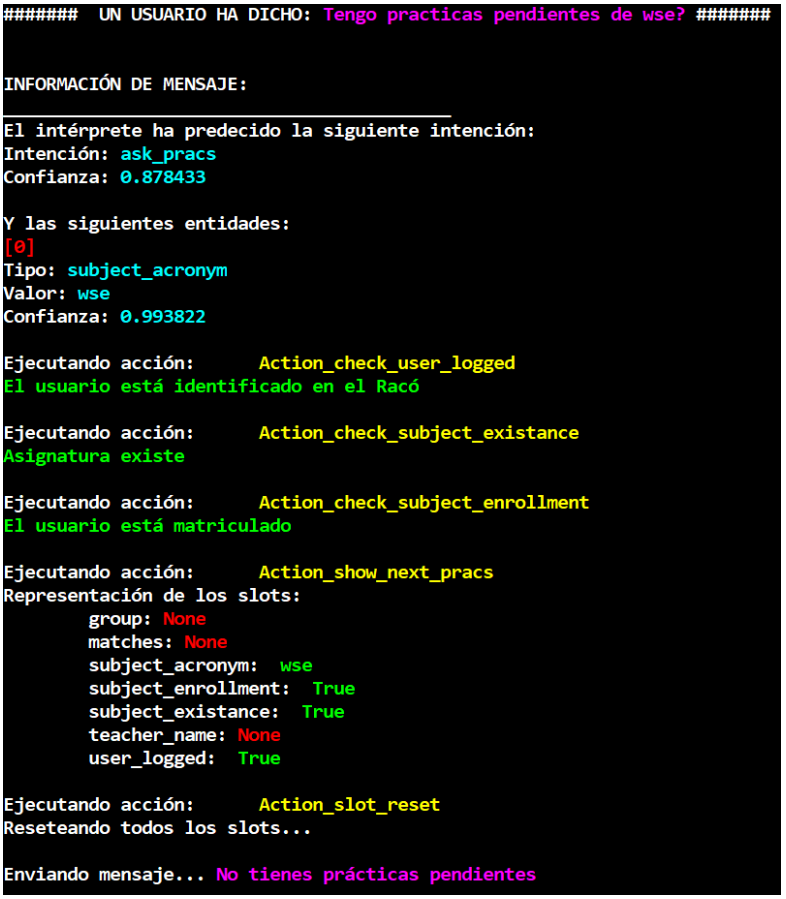

FIGURA 25: Procesado para una pregunta sobre prácticas.

Capítulo 10

Limitaciones

A lo largo del desarrollo del proyecto se han ido encontrando varias limitaciones, que se expondrán a continuación.

En primer lugar, la principal limitación encontrada durante el desarrollo del proyecto es la falta de documentación. Por un lado, el framework usado (Rasa), tiene una orientación de ser un producto fácil de incorporar a un sistema existente. Por lo que la documentación que existe está orientada a explicar como usar el framework como herramienta encargada del dialogo, pero no aporta demasiados detalles acerca de como funciona de forma interna. Ésto por ejemplo puede afectar en el proyecto cuando se quiere obtener información acerca de los resultados de entrenar un modelo, o para llevar a cabo ciertas modificaciones a los modelos para buscar mejoras.

En en segundo lugar, la API de Telegram tiene una limitación importante, y es que sólo admite la recepción de parámetros para el parámetro "start". Esto significa que cuando se requiere enviar un parámetro (como por ejemplo el código de autenticación de una API de terceros, como la API del Racó), se debe de recibir bajo el nombre de parámetro "start".

Esto ha provocado que en un paso del proceso de autenticación Oauth explicado en la sección 7.2.1 se deba enviar el código de autenticación mediante un mensaje con una URL, y no se pueda llevar a cabo de forma transparente como parámetro, pues la API del Racó devuelve el código de autenticación bajo el nombre de parámetro "code".

Otra limitación importante relacionada con la API del Racó es que es una API para atender a consultas, y no modificaciones. Por lo que la cantidad de opciones que habría para implementar un sistema más interactivo, es decir, donde hubiera la posibilidad de mantener diálogos más largos, se han visto reducidas.

Además, aunque la forma en la que está estructurada la API del Racó tiene sentido, hay bastantes elementos que no son coherentes.

Un ejemplo de ésto es que para referirse a las siglas de una asignatura dependiendo de la información que se está consultando se usan términos como: "id", "assig", "sigles", "codi_assig", etc, lo que hace confuso hacer ciertas consultas.

Otro problema de la API del Racó que a lo largo del desarrollo se ha observado, es que existe información no directamente accesible mediante las interfícies presentadas. Un ejemplo de esto es que en el listado de asignaturas que presenta la API (disponible en `https://api.fib.upc.edu/v2/assignatures/`) no existe ninguna asignatura llamada "VC" o "Visión por Computador". No obstante, si se accede a la dirección concreta (mediante la URL `https://api.fib.upc.edu/v2/assignatures/VC/`) se puede obtener la información acerca de la asignatura. Esto hace imposible navegar por la API para obtener datos porque no todos son accesibles.

Una limitación que también se ha comprobado es que cuando se decidió que el bot debía ser capaz de comunicarse tanto en catalán, castellano como en inglés se planteó probar de usar alguna API de traducción y sólo disponer de un modelo en castellano. Posteriormente se comprobó que estas APIs de traducción en su mayoría son de pago, y las que son de gratuitas no proporcionan buenas traducciones debido, en parte, al contexto tan específico en el que se enmarca el proyecto.

Otra limitación que aparece cuando el proyecto consiste en un chatbot suele ser la latencia. Para tener una buena experiencia de usuario se espera que el bot sea capaz de responder con rapidez. Este factor puede llegar a limitar la funcionalidad del bot, debido a que el uso de APIs en línea de terceros para obtener información añade latencia para responder al mensaje. Más concretamente, en el caso del proyecto se ha comprobado que esta latencia puede llegar a ser de unos 3 segundos.

Finalmente, en lo referente a los modelos, el framework de Rasa usa como principal componente de NLP la librería spaCy para Python. Esta librería dispone de recursos en castellano e inglés de entre los soportados por el bot. Por este motivo, en el tratamiento de mensajes en catalán se usan los recursos de spaCy en castellano. Esto significa, por ejemplo, que cuando se vectorizan los mensajes que están escritos en catalán (*word vectors*), se usan los vectores precargados en castellano.

Capítulo 11

Conclusiones

Hay tres tipos de conclusiones que se pueden extraer del proyecto: las de los modelos, las del proyecto en general, y conclusiones a nivel personal.

11.1. Conclusiones de los modelos

Las conclusiones que se pueden sacar de los modelos es que son fiables, eficaces, y funcionan como se espera en la mayoría de casos y para los tres idiomas.

Además, se ha comprobado que gracias a que los mensajes usados en aplicaciones de mensajería instantánea tienden a ser cortos, simples y directos, con características básicas de la frase, tales como word vectors (o context vectors en su defecto), es viable obtener intenciones y entidades de forma efectiva sin necesidad de disponer de muchos datos para entrenar los modelos.

A nivel de diálogo, enseñando al modelo unas cuantas interacciones que pueden suceder en la conversación, éste ha sido capaz de asimilar qué razonamiento se debe llevar a cabo para proporcionar la respuesta, teniendo en cuenta qué comprobaciones debe hacer en cada caso.

Así que los métodos escogidos para resolver las partes del problema relacionadas con el lenguaje natural han terminado siendo válidos, y efectivamente resuelven el problema planteado.

11.2. Conclusiones del proyecto

A nivel de proyecto y con respecto al problema que se pretende resolver, la aplicación desarrollada es capaz de proporcionar la información (que no necesariamente está disponible a través de la web de la facultad o del Racó) de forma muy rápida, mucho más que haciendo la búsqueda de la información de forma manual.

Además, la posibilidad de que el propio bot de forma automática pueda enviar mensajes a los usuarios con los avisos nuevos que se vayan colgando al Racó ha resultado ser una funcionalidad muy útil, y en la mayoría de ocasiones envía los avisos automáticos antes que el sistema de suscripción por correo electrónico que ofrece la facultad.

En relación a la plataforma de uso, utilizar Telegram ha sido una gran elección, puesto que la plataforma ofrece métodos muy fáciles y cómodos para interactuar con el bot, y además su uso es muy rápido. Eso significa que el bot es capaz de recibir una consulta y enviar su respuesta con poco tiempo de retardo. Además, Telegram es una aplicación bastante extendida por lo que muchos de los potenciales usuarios del bot posiblemente ya la tengan instalada.

Por último, infraestructura de Telegram para los comandos ha facilitado mucho la implementación del bot, evitando tener que manejar las opciones de idioma, identificación con la cuenta del Racó y otros mediante lenguaje natural.

11.3. Conclusiones personales

Personalmente, el proyecto me ha aportado muchas cosas. Primero, me ha hecho aprender mucho acerca del procesado del lenguaje natural y Chatbots en general. Éstos son dos temas que me interesan y que hasta el momento no había tenido la oportunidad de aprender.

Además, el proyecto me ha hecho aprender mucho acerca de cómo se debe planificar un proyecto, tanto temporal como económicamente y también a evaluar su sostenibilidad, lo cual es un aspecto que no he trabajado con mucha profundidad a lo largo del grado y son elementos importantes para un ingeniero.

Desgraciadamente, también me he visto obligado a aprender lo importante que es disponer de buena documentación. En muchas ocasiones a lo largo del proyecto la falta de documentación ha podido provocar potenciales retrasos en el desarrollo.

Capítulo 12

Futuras expansiones

Una de las formas en las que el proyecto puede expandirse es mediante la incorporación de nuevas intenciones y acciones a realizar. Esto en parte podría requerir una actualización de la API del Racó para que pueda proporcionar más información o, incluso, información extra en el caso de identificarse con la cuenta del Racó de un profesor. Actualmente las opciones para profesores y alumnos en la API son las mismas, pero podría ser interesante añadir información como qué alumnos han entregado prácticas de asignaturas que imparte el profesor.

Otra opción interesante sería evitar generar los datasets de forma automática y generarlos de forma manual. De tal manera se podría conseguir una mayor variedad en los datos, y consiguientemente, mejor efectividad al interpretar preguntas no cubiertas por los datos.

Otra idea sería transformar el bot implementado para que forme parte de un bot que pueda asistir a los visitantes de la página web de la facultad para facilitar el acceso a la información que se esté buscando.

También se podría aumentar el servicio de notificaciones añadiendo la posibilidad de que los estudiantes decidan si quieren que el bot les informe cuando falte un cierto tiempo para tener que entregar prácticas o cuando haya exámenes.

Sería interesante también añadir la posibilidad a la administración de la facultad, o a la delegación de estudiantes de mandar mensajes distribuidos a todos los usuarios del bot para mandar avisos importantes o anuncios.

Por último, aunque con el catalán, castellano e inglés la inmensa mayoría de los estudiantes podrían usar el bot, podría ser útil añadir más idiomas al bot.

Referencias

[1] Facultat d'Informàtica de Barcelona (FIB). *FIB API v2*. URL: https://api.fib.upc.edu/v2/ (visitado 03-03-2018).

[2] Universitat Politècnica de Catalunya (UPC). *Servei de Directori de la UPC*. URL: http://directori.upc.edu/directori/ (visitado 03-03-2018).

[3] Gobinda G. Chowdhury. "Natural language processing". En: *Annual Review of Information Science and Technology* 37.1 (2003), págs. 51-89. ISSN: 1550-8382. DOI: 10.1002/aris.1440370103.

[4] Jiwei Li y col. "Deep Reinforcement Learning for Dialogue Generation". En: *CoRR* abs/1606.01541 (2016). arXiv: 1606.01541. URL: http://arxiv.org/abs/1606.01541.

[5] Jiwei Li y col. "Adversarial Learning for Neural Dialogue Generation". En: *CoRR* abs/1701.06547 (2017). arXiv: 1701.06547. URL: http://arxiv.org/abs/1701.06547.

[6] Sepp Hochreiter y Jürgen Schmidhuber. "Long Short-Term Memory". En: *Neural Comput.* 9.8 (nov. de 1997), págs. 1735-1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

[7] Ilya Sutskever, Oriol Vinyals y Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". En: *CoRR* abs/1409.3215 (2014). arXiv: 1409.3215. URL: http://arxiv.org/abs/1409.3215.

[8] Kyunghyun Cho y col. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". En: *CoRR* abs/1406.1078 (2014). arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078.

[9] Oriol Vinyals y Quoc V. Le. "A Neural Conversational Model". En: *CoRR* abs/1506.05869 (2015). arXiv: 1506.05869. URL: http://arxiv.org/abs/1506.05869.

[10] Yoshua Bengio y col. "A Neural Probabilistic Language Model". En: *J. Mach. Learn. Res.* 3 (mar. de 2003), págs. 1137-1155. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=944919.944966.

[11] Huyen Nguyen, David Morales y Tessera Chin. "A Neural Chatbot with Personality". En: (2017). URL: https://web.stanford.edu/class/cs224n/reports/2761115.pdf.

[12] Dzmitry Bahdanau, Kyunghyun Cho y Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". En: *CoRR* abs/1409.0473 (2014). arXiv: 1409.0473. URL: http://arxiv.org/abs/1409.0473.

[13] Ismael Vallejo Ruiz. "Asistente virtual (Chatbot) para la web de la Facultad de Informática". En: (2015). URL: http://eprints.ucm.es/33443/ (visitado 04-03-2018).

[14] Nuance Communications. *Conversaciones automatizadas e inteligentes con un toque humano*. URL: https://www.nuance.com/es-es/omni-channel-customer-engagement/digital/virtual-assistant/nina.html (visitado 04-03-2018).

[15] Google. *Google Assistant - Your own personal Google*. URL: https://assistant.google.com/ (visitado 04-03-2018).

[16] Apple. *iOS - Siri - Apple*. URL: https://www.apple.com/es/ios/siri/ (visitado 04-03-2018).

[17] Microsoft. *Cortana. Tu asistente virtual y personal inteligente*. URL: https://www.microsoft.com/es-es/windows/cortana (visitado 04-03-2018).

[18] Amazon. *Amazon Alexa*. URL: https://developer.amazon.com/alexa (visitado 04-03-2018).

[19] Facultat d'Informàtica de Barcelona (FIB). *Facultat d'Informàtica de Barcelona*. URL: https://www.fib.upc.edu/ca (visitado 03-03-2018).

[20] Facultat d'Informàtica de Barcelona (FIB). *El Racó de la FIB*. URL: https://raco.fib.upc.edu/ (visitado 03-03-2018).

[21] Telegram Messenger LLP. *Bots: An introduction for developers*. URL: https://core.telegram.org/bots (visitado 03-03-2018).

[22] Telegram Messenger LLP. *Telegram Bot API*. URL: https://core.telegram.org/bots/api (visitado 03-03-2018).

[23] Rasa. *Rasa: Open source conversational AI for enterprise*. URL: https://rasa.com/ (visitado 08-03-2018).

[24] Rasa. *Rasa Stack: Open source conversational AI*. URL: https://rasa.com/products/rasa-stack/ (visitado 13-06-2018).

[25] spaCy. *Linguistic Features (Tokenization)- spaCy Usage Documentation*. URL: https://spacy.io/usage/linguistic-features#section-tokenization (visitado 14-05-2018).

[26] Erik F. Tjon Kim Sang. *Introduction to the CoNLL-2002 Shared Task: Language-Independent Named Entity Recognition*. URL: http://www.aclweb.org/anthology/W02-2024 (visitado 14-05-2018).

Anexo: Diagrama de Gantt

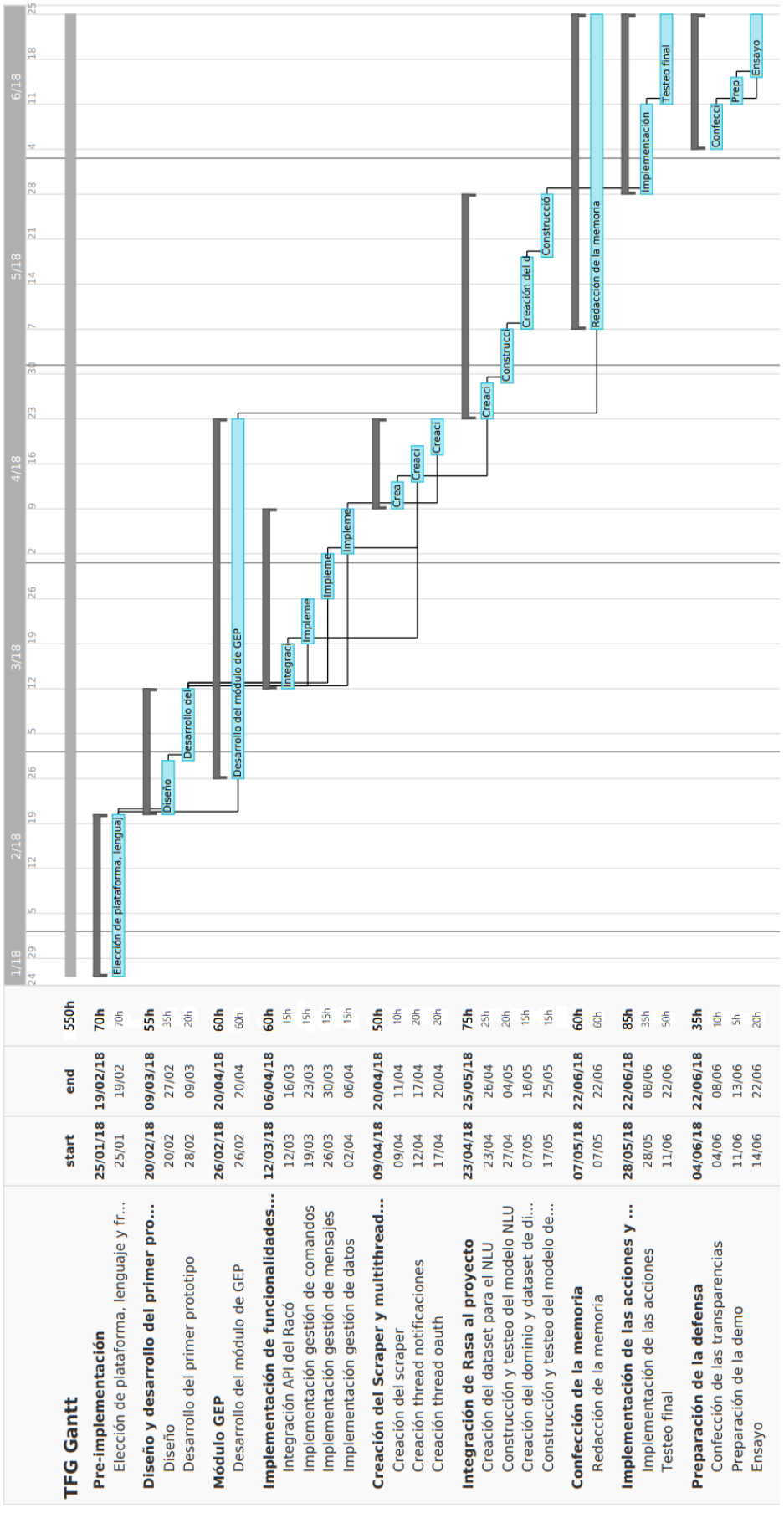

FIGURA 26: Diagrama de Gantt de la planificación.